# A Framework for Inducing Artificial Changes in Optimization Problems

Renato Tinós[a], Shengxiang Yang[*,b]

[a]*Department of Computing and Mathematics, FFCLRP, University of São Paulo (USP) 14040-901, Ribeirão Preto, SP, Brazil*
[b]*Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, U. K.*

**Abstract**

Environmental changes are traditionally considered intrinsic in evolutionary dynamic optimization. However, by ignoring that changes can instead be induced, we are ignoring that environmental changes can be eventually beneficial. To investigate the impact of artificial changes on the optimization speed up, we propose a framework for inducing artificial changes in any pseudo-Boolean or continuous optimization in this paper. Seven types of changes can be induced. Knowing when and how the changes occur allows us to design new strategies for evolutionary algorithms. Through computational experiments and illustrative examples, the impact of introducing changes in the optimization process is investigated. Experimental results indicate that changing the environments according to the proposed framework can lead to higher speed up, but not for all problems and change types. The best performance was obtained by change types that introduce plateaus and/or modify the gradient of regions of the fitness landscape around the current best solution. By doing this, the evolutionary dynamics is modified, eventually allowing the population to escape faster from local optima and reach new zones of the fitness landscape. Given a pseudo-Boolean or continuous optimization static problem, the proposed framework can be used to dynamically change the problem to speed up the optimization.

*Key words:* Evolutionary dynamic optimization, evolution strategies, genetic algorithms, artificial changes

[*]Corresponding author.
*Email addresses:* rtinos@ffclrp.usp.br (Renato Tinós), syang@dmu.ac.uk

## 1. Introduction

The expectation of applying *evolutionary algorithms* (EAs) to a wider range of complex problems in the future is high. Many of these problems are *dynamic optimization problems* (DOPs), i.e., problems that change over the optimization time. The investigation of *evolutionary dynamic optimization* (EDO) has become an important research trend for the evolutionary computation community Eiben and Smith (2015). Problems can change due to several factors, but the main consequence of such changes is the modification of the fitness landscape, which has been traditionally viewed as a serious complication in EDO. Once the environment changes, new optima should be found if the best solutions are affected by the fitness landscape modification. Many strategies have been developed in order to deal with the consequences of fitness landscape modifications Cruz et al. (2011); Nguyen et al. (2012).

The changes in the environments are traditionally considered *intrinsic* in EDO. In other words, the occurrence and types of changes are not controlled by the programmer. However, by ignoring that changes can instead be *induced*, we are ignoring that environmental changes can be eventually beneficial. The most obvious advantage of inducing changes in evolutionary computation is the possibility of an eventual speed up in the optimization process. Other advantages can exist. For example, changes can be induced during the optimization in order to search for robust solutions Fu et al. (2015). In experiments involving competition and cooperation, the programmer can control when and how the fitness landscape modifications are inserted Richter (2015); in order to obtain better solutions, the best ways of inducing competition or cooperation can be investigated. Despite the fact that the methodology developed in this paper can be applied to the investigation of other possible advantages, we will focus our attention on the investigation of the speed up of the optimization process by inducing environmental changes.

We propose investigating the impact of artificial changes on the optimization speed up from an EDO perspective. In order to do this, we propose a framework that allows us to induce artificial changes in *any* pseudo-Boolean or continuous optimization problem. Seven types of changes can be induced. This framework can be used to eventually speed up evolutionary optimization, given an instance of pseudo-Boolean or continuous problems. We investigate, through computational experiments and illustrative examples, the causes for the possible advantages of introducing changes in the optimization process. The fact that induced changes

(Shengxiang Yang)

in the fitness landscape can be beneficial for speeding up artificial evolution can appear counter-intuitive. However, we show that some of the induced changes proposed in this paper cause transformations in the fitness landscape that result in the creation of plateaus and/or modifications in the gradient of the regions around the current best solutions. Both modifications can speed up evolution: i) by eventually allowing the population to escape faster from local optima; and ii) by creating neutral zones that allow the population to reach new zones of the fitness landscape.

We believe that the investigation of artificially induced DOPs opens new research possibilities in EDO. In artificially induced DOPs, we can investigate the best ways of changing the fitness landscape in order to find faster the best optima for the static problem. Thus, researchers should investigate *when* and *how* to change the problem, which is not possible in intrinsic real-world DOPs. Besides, because we know beforehand when the changes occur, we can design new algorithms, strategies, and operators. In this paper, we illustrate this by investigating three strategies that explore this knowledge.

In intrinsic DOPs, researchers have no control over the environmental changes. In benchmark DOP generators, researchers have control over the environmental changes. However, in the benchmark DOP generators developed so far in the literature, the changes are not created in order to eventually help the optimization process. Instead, they are created in order to simulate intrinsic changes that occur in real-world problems. The proposed framework differs from benchmark DOP generators mainly because it generates artificially induced DOPs in order to speed up evolutionary optimization, given an instance of pseudo-Boolean or continuous optimization problems.

The rest of this paper is organized as follows. Related works are discussed in Section 2. The general framework for inducing changes is presented in Section 3. The same strategy is used to induce changes in pseudo-Boolean and continuous optimization problems, with differences in the types of changes. The types of changes for pseudo-Boolean and continuous optimization problems are respectively presented in Sections 4 and 5. Simple strategies for EAs applied to artificially induced DOPs are investigated in Section 6. We test our approaches in experiments with different EAs and optimization instances. The experimental results are presented in Section 7. Finally, the paper is concluded in Section 8.

## 2. Related Works

Some researchers investigated the possibility of speeding up artificial and biological evolution by inducing environmental changes Kashtan et al. (2007); Parter et al. (2008); Otwinowski et al. (2011); Tan and Gore (2012); Steinberg and Ostermeier (2016). In a paper titled "Varying environments can speed up evolution" Kashtan et al. (2007), Kashtan *et al.* investigated the impact on the evolution speed up of temporally changing the goals in five models: logic circuits, feed-forward logic circuits, feed-forward neural networks, feed-forward circuits, and artificial model for finding an RNA secondary structure. Two strategies were investigated: randomly varying goals and modularly varying goals. In the second strategy, which resulted in greater speed up, sub-goals of the cost function are frequently switched off and on during the optimization.

Changing the evaluation of solutions is also present in multi-objectivization, which is employed to make artificial evolutionary optimization easier Knowles et al. (2001). In multi-objectivization, sub-goals are incorporated to a single-objective problem, transforming it into a multi-objective problem. An example for transforming single-objective instances of the traveling salesman problem into multi-objective instances is presented in Knowles et al. (2001). In the standard traveling salesman problem, the shortest cycle traversing all the cities should be found. In the new problem, the set of cities is split into two or more subsets. The number of subsets defines the number of objectives in the problem. The $i$-th objective is to find the shortest tour traversing all the cities of the $i$-th subset.

The main hypothesis for explaining the possible advantages of inserting environmental changes in evolution is that they modify the evolution dynamics Tinós and Yang (2014), eventually allowing the population to escape from plateaus and to cross fitness valleys Kashtan et al. (2007). A similar hypothesis was investigated by Steinberg and Ostermeir Steinberg and Ostermeier (2016) in experiments with molecular evolution. Strategies for inducing environmental changes in the evolution of the antibiotic resistance gene *TEM-15* $\beta$-lactamase were experimentally tested. In a strategy which produced very good results, individuals with low antibiotic resistance were preferably selected in the initial steps of the experiment. Analyzing the evolution pathways, the researchers observed that a deleterious mutation in the first part of the experiment allowed to access a promising part of the sequence space, which was later explored. This sequence space region is rarely reached if the fitness landscape is not modified.

With the objective of investigating the impact of artificial changes on the optimization speed up, we proposed in Tinós and Yang (2016) a framework for induc-

ing six types of changes in *any* pseudo-Boolean problem. Other past works considered strategies for inducing environmental changes only in *specific* optimization problems Kashtan et al. (2007); Parter et al. (2008); Otwinowski et al. (2011). In the DOPs created by the framework proposed in Tinós and Yang (2016), environmental changes are regularly inserted in the optimization problem with the objective of speeding up evolution. The six types of changes considered in Tinós and Yang (2016) are based on those introduced in the DOP benchmark generator proposed in Tinós and Yang (2014).

Benchmark DOP generators are employed for testing operators, strategies, and algorithms specially designed for DOPs. The dynamism of the problem, e.g., the change frequency and severity, can be controlled when DOP generators are used. Benchmark DOP generators with different properties exist in the literature. Branke Branke (2001) and Morrison and De Jong Morrison (2004) proposed generators where a number of peaks are created in a base landscape. The peaks are then modified according to a given rule. The dynamism of the DOP is controlled by modifying the frequency and severity of changes.

A popular approach when creating a DOP is to change, during the optimization process, the components that define a static problem instance Mavrovouniotis et al. (2017). Benchmark DOP generators were created from different optimization problems. Examples are: dynamic traveling salesman problem Younes et al. (2003); Eyckelhof et al. (2002), dynamic vehicle routing problem Mavrovouniotis and Yang (2015), dynamic knapsack problem Rohlfshagen and Yao (2009), and dynamic job shop scheduling problem Zhou et al. (2009). Another example is the dynamic benchmark generator for permutation-encoded problems, which generates DOPs by swapping the nodes in the graph representation of an instance of a permutation problem Mavrovouniotis et al. (2012).

Another approach for creating DOPs is to change the fitness function of a problem. One of the most popular of such benchmark generators is the XOR DOP generator Yang (2003). The XOR DOP generator creates DOPs from any pseudo-Boolean stationary problem. In this generator, the fitness function of a given problem is evaluated at $\mathbf{x} \oplus \mathbf{m}$, where $\oplus$ is the XOR operator and $\mathbf{m}$ is a binary template modified every $\tau$ generations, instead of at position $\mathbf{x}$. The ideas behind the XOR DOP generator were used in benchmark DOP generators that allow to create DOPs from any stationary continuous optimization problem Li and Yang (2008); Tinós and Yang (2007). Other benchmark generators were recently proposed for continuous dynamic constrained Nguyen and Yao (2012) and multi-objective Helbig and Engelbrecht (2014) optimization.

Recently, ideas behind the XOR DOP generator were extended in Tinós and

Yang (2014) by considering other change types. The benchmark problem generator proposed in Tinós and Yang (2014) is capable of creating DOPs from any pseudo-Boolean optimization problems. Modifications in fitness landscapes of some pseudo-Boolean DOPs were theoretically investigated in order to create this problem generator. The investigated DOPs include problems created by the XOR DOP generator, three variants of the dynamic 0-1 knapsack problem Rohlfshagen and Yao (2009), the random dynamics NK-landscapes Eriksson and Olsson (2004), and the simulation of evolutionary robots with intermittent faults. Three types of transformations of the fitness landscapes were observed: 1) permutation of solutions; 2) copying a subset of decision variables or a subset of solutions; and 3) adding deviations to the fitness of a subset of solutions. Some of these transformations were not previously described in the EDO literature. Based on these transformations, six ways of generating DOPs were proposed. The work presented here extends the framework proposed in Tinós and Yang (2016), that is based on the generator proposed in Tinós and Yang (2014).

## 3. Framework for Inducing Artificial Changes

This paper considerably extends the work presented in Tinós and Yang (2016) by: 1) allowing the framework to generate modifications in regions of the fitness landscape around the current best solution. This is motivated because, doing so, the chance of escaping from local optima becomes higher; 2) introducing a new type of change and improving the framework by modifying the equations for the change types in order to equalize the impact of the change severity parameter; 3) extending the framework to continuous optimization problems; 4) adding more test problems, with a deeper analysis of the results; 5) introducing a new strategy that uses the knowledge about the occurrence of changes; 6) investigating, through computational experiments and illustrative examples, the causes for the possible advantages of introducing changes in the optimization process.

The general framework for inducing artificial changes is described in the following. The general framework can be applied to pseudo-Boolean and continuous optimization problems. However, the types of induced DOPs for pseudo-Boolean and continuous optimization problems are different.

We propose to transform a static problem $P$ with fitness function $f_P(\mathbf{x})$ into a DOP with fitness function as follows:

$$f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big) + \Delta f\big(g(\mathbf{x},e),e\big), \tag{1}$$

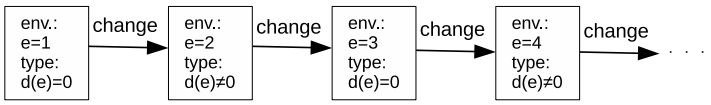


Figure 1: A DOP is seen as a sequence of static environments. In the proposed framework, the environments are equal to the static environment for problem $P$, i.e., $d(e) = 0$, when $\text{mod}(e,2) = 1$. When $\text{mod}(e,2) = 0$, the change type $d(e) \neq 0$ occurs.

where $\mathbf{x}$ is an $l$-dimensional candidate solution and the index of the environment is denoted by $e$. Each environment has a static fitness landscape Tinós and Yang (2014). In Eq. (1), the static fitness $f_P(.)$ for problem $P$ is evaluated at position $g(\mathbf{x},e)$. A deviation $\Delta f\big(g(\mathbf{x},e),e\big)$ is added to the evaluation $f_P(.)$. The framework allows to create seven different types of changes, by modifying the ways which $g(\mathbf{x},e)$ and $\Delta f\big(g(\mathbf{x},e),e\big)$ are chosen. The changes are of types 1.1, 1.2, 1.3, 2.1, 2.2, 2.3, and 3.1 (see Sections 4 and 5).

A created DOP can be seen as a sequence of environments generated by Eq. (1), as illustrated in Fig. 1. Each environment is associated with a variable $d(e)$, indicating the type of change. When $d(e) = 0$, the environment is equal to the static environment for problem $P$, i.e., $f(\mathbf{x},e) = f_P(\mathbf{x})$. Otherwise, the environment $e$ is created by transforming the problem $P$ using a change of type $d(e) \neq 0$.

The goal of the framework proposed here is to speed up evolution for a problem $P$. In this way, we propose switching between the fitness landscape for problem $P$ and a modified landscape. In other words, when $e$ is odd, i.e., $\text{mod}(e,2) = 1$, the $e$-th environment is equal to the static environment for problem $P$ and $d(e) = 0$. When $e$ is even, i.e., $\text{mod}(e,2) = 0$, the fitness of the environment is evaluated using Eq. (1) and $d(e) \in \{1.1, 1.2, 1.3, 2.1, 2.2, 2.3, 3.1\}$. The change period for each environment is fixed during the optimization. The change period is given by parameter $\tau \in \mathbb{N}^+$, which indicates the number of generations used by the algorithm for each environment. As we will see in next sections, another parameter, $\rho$, where $\{\rho \in \mathbb{R} | 0.0 \leq \rho \leq 1.0\}$, controls the severity of the change. The change frequency ($\tau$) and the change period ($\rho$) are the only parameters of the framework. While $\tau$ impacts on how often the environment changes, $\rho$ impacts on the severity of the fitness landscape modification. The changes introduced by the framework cause modifications in the fitness landscape around the current best solutions. This can help the population to escape from local optima. However, we are optimizing an instance of problem $P$; in this way, it is important to switch back to the static

Table 1: Types of changes in pseudo-Boolean optimization problems. The types are: 1.1. permutation of XOR type for all solutions; 1.2. permutation defined by decision variable exchanges; 1.3. permutation of XOR type for a subset of solutions; 2.1. duplicating decision variables; 2.2. moving solutions of a subset to a single random location; 2.3. moving solutions of a subset to the location of the best found solution; 3.1. adding deviation to the fitness of a subset of solutions.

| Change $d(e)$ | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big) + \Delta f\big(g(\mathbf{x},e),e\big)$ | Control Variables |
|---|---|---|
| 1.1 | $f(\mathbf{x},e) = f_P\big(\mathbf{x} \oplus \mathbf{m_{1.1}}(e)\big)$ | $\mathbf{m_{1.1}}(e) \in \mathbb{B}^l$: random template with $\lfloor \rho l \rfloor$ ones |
| 1.2 | $f(\mathbf{x},e) = f_P\big(\mathbf{m_{1.2}}(\mathbf{x},e)\big)$ | $\mathbf{m_{1.2}}(\mathbf{x},e) \in \mathbb{B}^l$: obtained by randomly exchanging $\lfloor 2\rho l \rfloor$ elements of $\mathbf{x}$ |
| 1.3 | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big)$, $g(\mathbf{x},e) = \begin{cases} \mathbf{x} \oplus \mathbf{m_{1.3}}(e), & \text{if } \mathbf{x} \in \mathbf{s}(e) \\ \mathbf{x}, & \text{otherwise} \end{cases}$ | $\mathbf{s}(e)$: random hyperplane with $\omega$ fixed bits and where $\mathbf{b}(e-1) \in \mathbf{s}(e)$<br>$\mathbf{m_{1.3}}(e) \in \mathbf{s}(e)$: random binary template with $\lfloor \rho l \rfloor$ ones |
| 2.1 | $f(\mathbf{x},e) = f_P\big(\mathbf{m_{2.1}}(\mathbf{x},e)\big)$ | $\mathbf{m_{2.1}}(\mathbf{x},e) \in \mathbb{B}^l$: obtained by randomly duplicating $\lceil \frac{\rho l}{2} \rceil$ elements of $\mathbf{x}$ |
| 2.2 | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big)$, $g(\mathbf{x},e) = \begin{cases} \mathbf{m_{2.2}}(e), & \text{if } \mathbf{x} \in \mathbf{s}(e) \\ \mathbf{x}, & \text{otherwise} \end{cases}$ | $\mathbf{s}(e)$: random hyperplane with $\omega$ fixed bits and where $\mathbf{b}(e-1) \in \mathbf{s}(e)$<br>$\mathbf{m_{2.2}}(e) \in \mathbf{s}(e)$: random binary solution |
| 2.3 | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big)$, $g(\mathbf{x},e) = \begin{cases} \mathbf{b}(e-1), & \text{if } \mathbf{x} \in \mathbf{s}(e) \\ \mathbf{x}, & \text{otherwise} \end{cases}$ | $\mathbf{s}(e)$: random hyperplane with $\omega$ fixed bits and where $\mathbf{b}(e-1) \in \mathbf{s}(e)$ |
| 3.1 | $f(\mathbf{x},e) = f_P(\mathbf{x}) + \Delta f\big(g(\mathbf{x},e),e\big)$, $\Delta f\big(g(\mathbf{x},e),e\big) = \begin{cases} a(e), & \text{if } \mathbf{x} \in \mathbf{s}(e) \\ 0, & \text{otherwise} \end{cases}$ | $\mathbf{s}(e)$: random hyperplane with $\omega$ fixed bits and where $\mathbf{b}(e-1) \in \mathbf{s}(e)$<br>$a(e) \in \mathbb{R}$: random uniform deviation in the range $[-\rho f_r, +\rho f_r]$ |

fitness landscape in order to optimize the original instance. We do this several times, i.e., to introduce a change that can allow the population to escape from local optima and then to let the population evolve in the original fitness landscape.

## 4. Generating Dynamic Pseudo-Boolean Optimization Problems

In a static pseudo-Boolean optimization problem $P$, the fitness function is $f_P(\mathbf{x}) \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{B}^l$ is the $l$-dimensional candidate solution vector. A static pseudo-Boolean optimization problem $P$ is transformed into a DOP by changing $g(\mathbf{x},e)$ and $\Delta f\big(g(\mathbf{x},e)$ in Eq. (1). Table 1 shows how $g(\mathbf{x},e)$ and $\Delta f\big(g(\mathbf{x},e)$ are generated for each type of change $d(e) \neq 0$. The control variables are defined when the problem changes, i.e., in the transitions between an environment with $d(e) = 0$ and an environment with $d(e) \neq 0$. In the following, we first give some general definitions and then describe each type of change.

For Changes 1.3, 2.2, 2.3, and 3.1, a candidate solution $\mathbf{x}$ is affected by a change only if $\mathbf{x} \in \mathbf{s}(e)$. The hyperplane $\mathbf{s}(e)$ defines a subset of solutions of the search space $\mathbb{B}^l$. All solutions in the subset defined by $\mathbf{s}(e)$ share a subset of fixed bits. The hyperplane, or schema, $\mathbf{s}(e)$ is an $l$-dimensional string composed of digits 0, 1 and * ("do not care"). All solutions $\mathbf{x} \in \mathbf{s}(e)$ have in common $\omega$ fixed bits, i.e., elements that are not "do not care" digits. The number of fixed bits, $\omega$,

Table 2: The order of template $\mathbf{s}(e)$ for different values of $\rho$ (Eq. 2). Here, we consider that $l > 20$. The percentage of solutions affected by Changes 1.3, 2.2, 2.3, and 3.1 is equal to $100\rho/2$

| $\rho$ | 1.0 | 0.5 | 0.3 | 0.2 | $10^{-1}$ | $10^{-2}$ | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ | $10^{-6}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $\omega$ | 1 | 2 | 2 | 3 | 4 | 7 | 10 | 14 | 17 | 20 |

is the order of $\mathbf{s}(e)$. The motivation for using $\mathbf{s}(e)$ is the fact that some changes in real problems affect only candidate solutions that share specific combinations of decision variables Tinós and Yang (2014).

Unlike the framework proposed in Tinós and Yang (2014), the framework proposed here considers that the DOPs created by Changes 1.3, 2.2, 2.3, and 3.1 have the time-linkage property. When a DOP has the time-linkage property, the population path generated by the optimization algorithm influences the future dynamics of the problem Nguyen et al. (2012). Here, the best solution found by the algorithm in environment $e-1$, given by $\mathbf{b}(e-1)$, influences environment $e$ when $d(e) \neq 0$. We consider that the best solution $\mathbf{b}(e-1)$ belongs to the subset defined by hyperplane $\mathbf{s}(e)$. In practice, $\mathbf{s}(e)$ is randomly generated ensuring that $\mathbf{b}(e-1) \in \mathbf{s}(e)$, i.e., we randomly choose $\omega$ decision variables of $\mathbf{b}(e-1)$ as the fixed bits of $\mathbf{s}(e)$. In the framework proposed in this paper, the order of $\mathbf{s}(e)$ is:

$$\omega = \begin{cases} \lfloor -\log_2 \frac{\rho}{2} \rfloor, & \text{if } \lfloor -\log_2 \frac{\rho}{2} \rfloor \leq l \\ l, & \text{otherwise} \end{cases} \tag{2}$$

where $\rho$ controls the change severity. For Changes 1.3, 2.2, 2.3, and 3.1, the fraction of the search space affected by a change is equal to $\rho/2$. Table 2 shows the order of $\mathbf{s}(e)$ for some values of $\rho$.

### *4.1. Changes 1.1, 1.2, and 1.3: Permutation of Solutions*

The three first changes (1.1, 1.2, and 1.3) generate permutations in the solutions of the search space. For Change 1.1, the permutation is generated by $g(\mathbf{x}, e) = \mathbf{x} \oplus \mathbf{m_{1.1}}(e)$, where $\mathbf{m_{1.1}}(e) \in \mathbb{B}^l$ is a template randomly generated for each environment $e$ when $d(e) = 1.1$. The number of ones in $\mathbf{m_{1.1}}(e)$ is $\lfloor \rho l \rfloor$. More ones in the template means that more bits of $\mathbf{x}$ are switched, i.e., the Hamming distance between $g(\mathbf{x}, e)$ and $\mathbf{x}$ increases with the severity parameter $\rho$. Change 1.1 moves the solution $\mathbf{x}$ to a new location in the fitness landscape according to the XOR operation. Change 1.1 creates a special type of DOP with permutation Tinós and Yang (2014).

For Change 1.2, the new position $g(\mathbf{x}, e)$ is obtained by exchanging the decision variables of $\mathbf{x}$, i.e., $g(\mathbf{x}, e) = \mathbf{m_{1.2}}(\mathbf{x}, e)$, where $\mathbf{m_{1.2}}(\mathbf{x}, e) \in \mathbb{B}^l$ defines a

**Algorithm 1** Change Type 2.1.

1: $\mathbf{m_{2.1}} = \mathbf{x}$;
2: **for** $i$=1:$\lfloor \frac{\rho l}{2} \rfloor$ **do**
3: $m_{2.1}\big(random(1,\ldots,l)\big) = m_{2.1}\big(random(1,\ldots,l)\big)$;
4: **end for**

permutation of $\mathbf{x}$. The number of bits that are exchanged is $\lfloor 2\rho l \rfloor$. For each environment $e$, when $d(e) = 1.2$, a vector $\mathbf{m_{1.2}}(.)$ specifying a permutation is randomly generated. While all solutions of the original search space are affected by Change 1.1, the number of solutions affected by Change 1.2 varies from $2^{l-1}$ to $2^l - 2$ for $\rho > 0$ Tinós and Yang (2014). Some solutions of the search space are not affected by Change 1.2. For example, solutions $\mathbf{0}$ and $\mathbf{1}$ are not affected because any permutation of decision variables in such solutions does not change the solutions. Both Changes 1.1 and 1.2 preserve the neighborhood relations among the solutions of the search space. When a change occurs, solutions are moved to new locations. However, if two solutions are neighbors in the fitness landscape for $d(e) = 0$, they will remain as neighbors after changes $d(e) = 1.1$ and $d(e) = 1.2$. If, instead of moving the search space, the current solutions are moved in the same way, the effect in the optimization process is the same Tinós and Yang (2014).

On the other hand, Change 1.3 does not preserve the neighborhood relations. Only a subset of solutions of the search space is affected by Change 1.3. When a solution is affected by the change, the solution is permuted in the same way as in Change 1.1. In this way, besides generating the random template for the XOR transformation, here given by $\mathbf{m_{1.3}}(e)$, the hyperplane $\mathbf{s}(e)$ that defines the subset of solutions affected by the change is also defined for each environment $e$ when $d(e) = 1.3$. The hyperplane $\mathbf{s}(e)$ is randomly generated with $\omega$ fixed positions (Eq. (2)), ensuring that both $\mathbf{b}(e-1)$ and $\mathbf{m_{1.3}}(e)$ are in the subset defined by $\mathbf{s}(e)$. The same occurs for Changes 2.2, 2.3, and 3.1.

### *4.2. Changes 2.1, 2.2, and 2.3: Copying Solutions and Decision Variables*

Change 2.1 moves a candidate solution $\mathbf{x}$ to another location $\mathbf{m_{2.1}}(\mathbf{x}, e)$, that is obtained by randomly duplicating $\lceil \frac{\rho l}{2} \rceil$ elements of $\mathbf{x}$. Algorithm 1 shows how $\mathbf{m_{2.1}}(\mathbf{x}, e)$ is generated. The number of candidate solutions of the search space affected by Change 2.1 varies from $2^{l-1}$ to $2^l - 2$ for $\rho > 0$.

Instead of duplicating the decision variables, Changes 2.2 and 2.3 move all candidate solutions $\mathbf{x} \in \mathbf{s}(e)$ to a single location. As we will see in Section 5, Changes 2.2. and 2.3 produce plateaus on the fitness landscape. The only difference between Changes 2.2 and 2.3 is in the way they generate the new location for

$\mathbf{x}$. When Change 2.2 is applied, the new location $g(\mathbf{x},e)$ of $\mathbf{x} \in \mathbf{s}(e)$ is defined by a random solution, $\mathbf{m_{2.2}}(e)$, which is inside the subset defined by hyperplane $\mathbf{s}(e)$. When Change 2.3 is applied, the new location $g(\mathbf{x},e)$ of $\mathbf{x} \in \mathbf{s}(e)$ is the location of the best solution found in environment $e-1$, i.e., $g(\mathbf{x},e) = \mathbf{b}(e-1)$. Change 2.3 was not present in the DOP generator proposed in Tinós and Yang (2014) and in the framework for inducing DOPs proposed in Tinós and Yang (2016).

### *4.3. Change 3.1: Adding Deviations to the Fitness*

The previous changes types move the candidate solutions to new locations of the search space. On the other hand, Change 3.1 causes modifications in the fitness landscape by adding a deviation to the fitness of a subset of candidate solutions. If $\mathbf{x} \in \mathbf{s}(e)$, a random deviation $a(e) \in \mathbb{R}$, uniformly generated in the range $[-\rho f_r, +\rho f_r]$ for each environment $e$, is added to the fitness of the candidate solution. In order to generate the range $[-\rho f_r, +\rho f_r]$, $f_r$ is computed as the difference between the maximum and mean fitness in the initial population if this difference is equal to or greater than 0.1. Otherwise, $f_r$ is equal to the absolute value of the maximum fitness found in the initial population.

## 5. Generating Dynamic Continuous Optimization Problems

In a static continuous optimization problem $P$, the fitness function is $f_P(\mathbf{x}) \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^l$ is an $l$-dimensional candidate solution vector. Changes for continuous optimization problems are based on those for pseudo-Boolean optimization problems (Section 4). Table 3 shows how $g(\mathbf{x},e)$ and $\Delta f\big(g(\mathbf{x},e)$ are generated in Eq. (1) for each type of change $d(e)$ in continuous optimization problems.

Changes 1.3, 2.2, 2.3, and 3.1 affect only a subset of the search space. In order to define the subset of solutions affected by the changes, a vector $\mathbf{u}(e) \in \mathbb{B}^l$ with $\omega$ ones is randomly generated for each environment $e$. The number $\omega$ of ones in $\mathbf{u}(e)$ is given by Eq. (2). A candidate solution $\mathbf{x}$ is affected by a change of type 1.3, 2.2, 2.3, and 3.1 if:

$$|x_i - b_i(e-1)|u_i \leq \delta x, \forall i = 1,\ldots,l, \tag{3}$$

where $\mathbf{b}(e-1)$ is the best solution found in environment $e-1$ and $\delta x = 0.05|x^{sup} - x^{inf}|$. Variables $x^{sup}$ and $x^{inf}$ respectively indicate the upper and lower bounds for the continuous decision variables (if the bounds are not known, they can be defined *a priori* as large numbers).

Table 3: Types of changes in continuous optimization problems. The types are: 1.1. rotating all solutions according to a rotation matrix; 1.2. rotating the axes of the search space; 1.3. rotating a subset of solutions according to a rotation matrix; 2.1. duplicating decision variables; 2.2. moving solutions of a subset to a single random location; 2.3. moving solutions of a subset to the location of the best found solution; 3.1. adding deviation to the fitness of a subset of solutions.

| Change $d(e)$ | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big) + \Delta f\big(g(\mathbf{x},e),e\big)$ | Control Variables |
|---|---|---|
| 1.1 | $f(\mathbf{x},e) = f_P\big(\mathbf{A}(e)\mathbf{x}_s\big)$ | $\mathbf{A}(e) \in \mathbb{R}^{l\times l}$: random rotation matrix |
| 1.2 | $f(\mathbf{x},e) = f_P\big(\mathbf{m_{1.2}}(\mathbf{x},e)\big)$ | $\mathbf{m_{1.2}}(\mathbf{x},e) \in \mathbb{R}^l$: obtained by exchanging $\lfloor 2\rho l \rfloor$ elements of $\mathbf{x}$ |
| 1.3 | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big)$, $g(\mathbf{x},e) = \begin{cases} \mathbf{A}(e)\mathbf{x}_s, \text{ if } \lvert x_i - b_i(e-1)\rvert u_i \le \delta x, \forall i \\ \mathbf{x}_s, \quad \text{otherwise} \end{cases}$ | $\mathbf{u}(e) \in \mathbb{B}^l$: random vector with $\omega$ ones<br>$\mathbf{A}(e) \in \mathbb{R}^{l\times l}$: random rotation matrix |
| 2.1 | $f(\mathbf{x},e) = f_P\big(\mathbf{m_{2.1}}(\mathbf{x},e)\big)$ | $\mathbf{m_{2.1}}(\mathbf{x},e) \in \mathbb{R}^l$: obtained by randomly duplicating $\lceil \frac{\rho l}{2} \rceil$ elements of $\mathbf{x}$ |
| 2.2 | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big)$, $g(\mathbf{x},e) = \begin{cases} \mathbf{m_{2.2}}(e), \text{ if } \lvert x_i - b_i(e-1)\rvert u_i \le \delta x, \forall i \\ \mathbf{x}, \quad \text{otherwise} \end{cases}$ | $\mathbf{u}(e) \in \mathbb{B}^l$: random vector with $\omega$ ones<br>$m_{2.2i}(e) = \begin{cases} b_i(e-1), & \text{if } u_i = 1 \\ \text{rand}(x^{inf}, x^{sup}), & \text{if } u_i = 0 \end{cases}$ |
| 2.3 | $f(\mathbf{x},e) = f_P\big(g(\mathbf{x},e)\big)$, $g(\mathbf{x},e) = \begin{cases} \mathbf{b}(e-1), \text{ if } \lvert x_i - b_i(e-1)\rvert u_i \le \delta x, \forall i \\ \mathbf{x}, \quad \text{otherwise} \end{cases}$ | $\mathbf{u}(e) \in \mathbb{B}^l$: random vector with $\omega$ ones |
| 3.1 | $f(\mathbf{x},e) = f_P(\mathbf{x}) + \Delta f\big(g(\mathbf{x},e),$ $\Delta f\big(g(\mathbf{x},e) = \begin{cases} a(e), \text{ if } \lvert x_i - b_i(e-1)\rvert u_i \le \delta x, \forall i \\ 0, \quad \text{otherwise} \end{cases}$ | $\mathbf{u}(e) \in \mathbb{B}^l$: random vector with $\omega$ ones<br>$a(e) \in \mathbb{R}$: random uniform deviation in the range $[-\rho f_r, +\rho f_r]$ |

### *5.1. Changes 1.1, 1.2, and 1.3: Rotation of Solutions*

Based on the analysis of permutation changes in pseudo-Boolean optimization problems, the authors in Tinós and Yang (2007) proposed the generation of continuous DOPs by rotating the candidate solutions in the search space. This idea is used to generate Changes 1.1 and 1.3. After scaling the decision variables between -1 and 1, the rotation is given by $g_s(\mathbf{x},e) = \mathbf{A}(e)\mathbf{x}_s$, where $\mathbf{x}_s$ represents the scaled solution vector. Matrix $\mathbf{A}(e)$ is a rotation matrix obtained by the successive multiplication of simple planar rotation matrices with angle $\rho\pi$. The order for the plane rotations, i.e., the order for the multiplication of the simple rotation matrices, is random. The parameter $\rho$ controls the severity by directly influencing the degree of the rotations. The algorithm to compute the rotation $\mathbf{A}(e)$ is given in Tinós and Yang (2007). After rotating the scaled solution $\mathbf{x}_s$ in order to obtain $g_s(\mathbf{x},e)$, the new location $g(\mathbf{x},e)$ is obtained by applying the inverse of the scale function in $g_s(\mathbf{x},e)$.

While all solutions of the search space are rotated by Change 1.1, only those in the subset defined by Eq. (3) are rotated by Change 1.3. Change 1.2 is similar to that one presented for pseudo-Boolean optimization problems. Change 1.2 acts

by permuting the decision variables, which is similar to rotating the axes of the search space Tinós and Yang (2007).

### *5.2. Changes 2.1, 2.2, and 2.3: Copying Solutions and Decision Variables*

Change 2.1 is similar to that one used in pseudo-Boolean optimization problems. In other words, $\lfloor \frac{\rho l}{2} \rfloor$ real decision variables are randomly copied to other positions. Changes 2.2 and 2.3 are also similar to those for pseudo-Boolean optimization problems. They affect a subset of solutions generated in the same way as for Change 1.3, i.e., only if $|x_i - b_i(e-1)|u_i \leq \delta x, \forall i = 1, \ldots, l$. While the solutions are moved to $\mathbf{b}(e-1)$ for Change 2.3, they are equal to a random solution generated inside the subset defined by $|x_i - b_i(e-1)|u_i \leq \delta x, \forall i = 1, \ldots, l$ for Change 2.2.

### *5.3. Change 3.1: Adding Deviations to the Fitness*

When a solution is affected by Change 3.1, its fitness is modified by adding a random deviation $a(e) \in \mathbb{R}$, uniformly generated in the range $[-\rho f_r, +\rho f_r]$ for each environment $e$. The subset of solutions affected by Change 3.1 is defined in the same way as in Changes 1.3, 2.2, and 2.3. The random deviation $a(e) \in \mathbb{R}$ is generated in the same way as in pseudo-Boolean optimization problems.

### *5.4. Visualization of the Effects of Changes on the Fitness Landscape*

The effects of changes in the fitness of solutions can be observed in Fig. 2, which shows the fitness of the solutions, before and after the changes, for two-dimensional continuous optimization problem $f_{12}$. Problem $f_{12}$ will be presented in Section 7. However, the fitness of solutions for this problem with $l = 2$ can be observed on the top, left graph of Fig. 2. The remaining graphs illustrate examples of the transformations of problem $f_{12}$ by each type of change. For all changes, the change severity parameter, $\rho$, is 0.5. The landscapes for Changes 1.1 and 1.2 are obtained by the rotation of the search space. For Change 1.1, the fitness landscape modification is obtained by rotating all solutions by $0.5\pi$ radians, while the modification is obtained by permuting the decision variables, i.e., the axes of the search space, for Change 1.2. One can observe that the neighborhood relations are not changed after the transformations. This does not occur for Change 1.3, where only solutions of the subset defined by $|x_i - b_i(e-1)|u_i \leq \delta x, \forall i = 1, \ldots, l$ are rotated by $0.5\pi$ radians.

The idea of changing only a subset of solutions is also present in Changes 2.2, 2.3, and 3.1. The size of the subset of solutions affected by the changes is the same; however, the location depends on the best solution found for environment

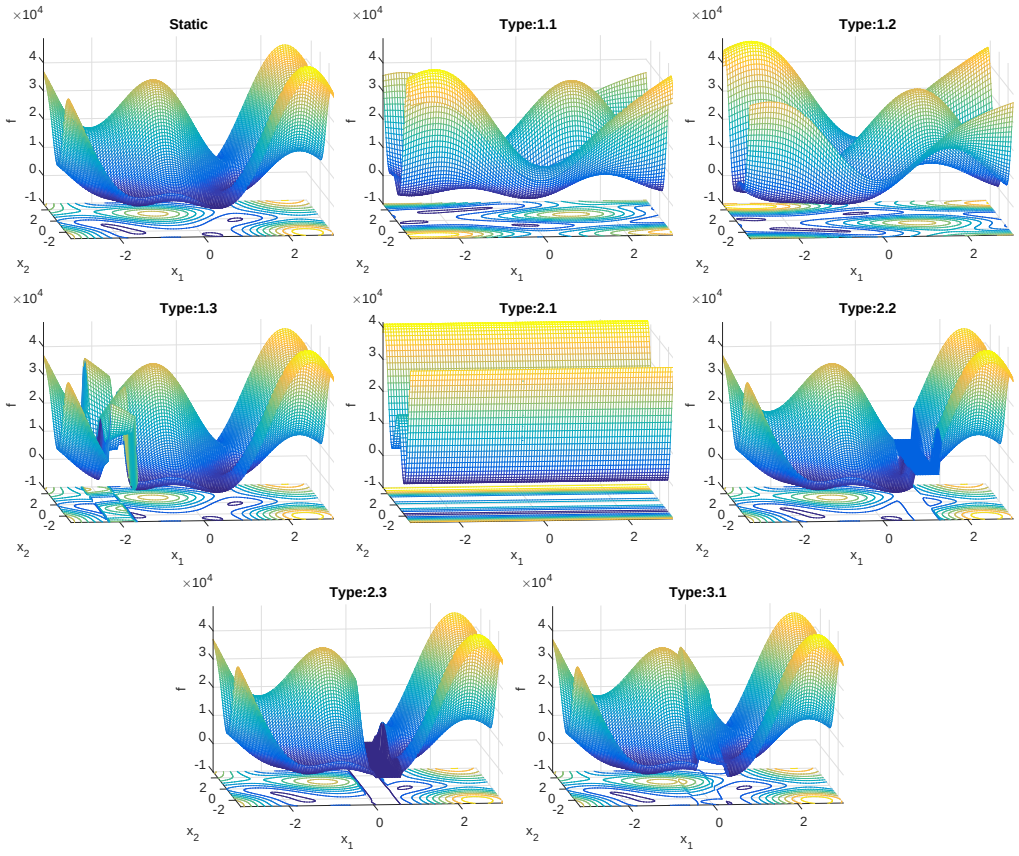


Figure 2: Fitness for static and dynamic instances of problem $f_{12}$ with $l = 2$ and $\rho = 0.5$. Each graph shows the fitness in an environment created a different change type.

$e-1$, i.e., $\mathbf{b}(e-1)$. Plateaus are formed in the landscapes affected by Changes 2.2 and 2.3. While the height of the plateau is equal to the fitness of $\mathbf{b}(e-1)$ for Change 2.3, it is equal to the fitness of a solution randomly chosen inside this plateau for Change 2.2.

The fitness landscape is drastically modified by Change 2.1. In this case, the fitness for all solutions that have the same decision variable at axis $x_1$ for $\mathbf{b}(e-1)$ are copied to all the other points at axis $x_1$. We can observe that the fitness is preserved if only the decision variable $x_1$ is modified. As a consequence, the contour curves are parallel to axis $x_1$.

The gradient of the region around $\mathbf{b}(e-1)$ is modified for Changes 1.1, 1.2, 1.3, 2.1 and 3.1. While the landscape is modified by rotation for Changes 1.1, 1.2, and 1.3, it is modified by adding a deviation for a subset of solutions for Change

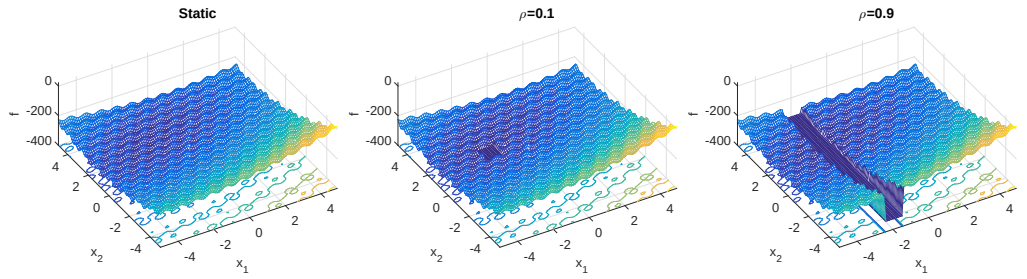


Figure 3: Fitness for static and dynamic instances of problem $f_{10}$ with $l = 2$. Each graph shows the fitness in an environment created by Change Type 2.3 with a different value of $\rho$.

3.1. The effects of changes in modifying the landscapes, and particularly the basis of attraction around local optima, can be observed in the contour curves in Fig. 2.

The effect of changing the change severity parameter $\rho$ can be observed in Fig. 3, where the fitness landscape for problem $f_{10}$ is modified by Change 2.3 with two values of $\rho$: 0.1 and 0.9. When defining the size of the subset of solutions affected by Change 2.3, the number of fixed decision variables, $\omega$, for $\rho = 0.1$ is 2, while for $\rho = 0.9$, $\omega = 1$ (Eq. (2)). As a consequence, the subset of solutions affected by the change is much larger for $\rho = 0.9$. As commented before, the percentage of solutions affected by the changes increases when $\rho$ increases.

## 6. Strategies for EAs Applied to Artificially Induced DOPs

Unlike intrinsic DOPs, we know beforehand when the changes occur in induced DOPs. Thus, new strategies and operators can be designed using this information. Also, the information about the types of changes can be used by specifically designed strategies and operators. In order to show that the information about the changes can be used to design new strategies, three simple strategies for EAs are proposed here, which are variants of strategies used in EDO Cobb and Grefenstette (1993); Yang (2008).

The first two strategies are based on immigrant strategies, where part of the population is replaced by random or stored solutions Yang (2008). Here, instead of replacing part of the population by immigrants in every iteration of the algorithm, the replacement occurs only when $d(e) = 0$, i.e., when the fitness landscape is equal to that of the static problem $P$. Two immigrant strategies are tested. In the Random Immigrants ($RIs$) strategy, 20% of the population is replaced by randomly generated individuals. In the Memory Immigrants ($MIs$), 10% of the population is replaced by individuals stored in a memory population. As the objective is to

optimize solutions for the static problem $P$, the memory population is composed of the best individuals found in environments where $d(e) = 0$. When a change $d(e) \neq 0$ occurs, the best individual is stored in the memory population. As the environments with changes and without changes are alternating, the individual stored when a change $d(e) \neq 0$ occurs is the best individual found in environment $e-1$ (with $d(e) = 0$). Because all the individuals in the memory population are generated in environments with the same fitness landscape, it is not necessary to re-evaluate them when they are re-introduced into the main population. The memory and main populations have the same size. When the maximum size is reached for the memory population, the new individual replaces a random individual, with exception for the best individual stored in the memory population.

The third strategy (*mInc*) is similar to the hypermutation strategy, where the mutation paremeter is increased whenever a change is detected Cobb and Grefenstette (1993). Here, instead of increasing the mutation parameter only when the change is detected, it is increased also one generation before the change. This is possible because we know beforehand when the changes will occur in induced DOPs. The three strategies are presented in Algorithm 2. The framework for inducing artificial changes in optimization problems, proposed in this work, can also be seen in Algorithm 2. The framework and strategies proposed in this paper do not depend on the type of EAs.

## 7. Experiment Study

Experiments with pseudo-Boolean and continuous optimization problems were conducted in order to test the proposed framework and strategies[1].

### *7.1. Experimental Design*

Genetic algorithms (GAs) are used for optimizing the pseudo-Boolean problems, while evolution strategies (ESs) are used in continuous optimization problems. The two algorithms have been chosen because they present many characteristics that are common to other population-based meta-heuristics. We test the immigrant strategies (*MI* and *RI*) in the GAs and the *mInc* strategy is tested in the ESs. We want to investigate the hypothesis that changing the environment, in the way proposed in Section 3, can be beneficial to the optimization of solutions for a given problem $P$. We also want to investigate whether the strategies proposed

[1] All experiments were executed in a server with 2 processors Intel Xeon E5-2620 v2 (15 MB Cache, 2.10 GHz) and 32 GB of RAM.

**Algorithm 2** EA with the framework of inducing artificial changes of type $c = \{1.1, 1.2, 1.3, 2.1, 2.2, 2.3, 3.1\}$

```
1: Initialize population P;
2: g = i = 0;
3: e = 1;
4: d(e) = 0;
5: while not (stopping criterion) do
6:    g = g + 1;
7:    if strategy = mInc and g + 1 − i ≥ τ then
8:       Increase mutation parameter;
9:    end if
10:   if g − i ≥ τ then
11:      i = g;
12:      e = e + 1;
13:      if mod(e, 2) = 1 then
14:         d(e) = 0;
15:         if strategy = MI then
16:            Insert immigrants in P from memory population M;
17:         end if
18:      else
19:         d(e) = c;
20:         if strategy = MI then
21:            Update memory population M;
22:         end if
23:      end if
24:      Define the control variables for change type d(e);
25:      if strategy = RI then
26:         Insert random immigrants in P;
27:      end if
28:      Evaluate the individuals in P considering change type d(e);
29:   end if
30:   Selection and Reproduction;
31:   Evaluate the individuals in P considering change type d(e);
32: end while
```

in Section 6 are beneficial in induced DOPs. As we want to investigate whether the possible benefit of changing the environment is due only to an increase in the diversity of the population when the problem changes, experiments where the population of the algorithms is equal to 1 are also presented.

All changes, except for Change 3.1, modify the fitness landscape by evaluating the solution $\mathbf{x}$ in a different location, $g(\mathbf{x}, e)$. In this way, we also investigate whether changing the problem according to the proposed framework with change frequency $\tau$ is equivalent to adopting a search strategy where the solutions are randomly moved ("kicked") to new positions every $\tau$ generations. When the population size is equal to 1, this strategy is similar to an iterated local search. When the kick size is equal to 1, this strategy is similar to the restart approach. For the pseudo-Boolean optimization problems, when a solution is "kicked", bit flip is applied to random decision variables $\lceil k_s l \rceil$ times, where $k_s$ is the kick size. For the continuous optimization problems, the kick is randomly generated in a hyper-sphere with center in the solution before the kick. The radius of the hyper-sphere is equal to the kick size multiplied by the length of the decision variables range used to generate the initial population.

We compare the performance of algorithms in dynamic instances generated using the proposed framework and in static instances. For static instances, we consider runs with different "kick" sizes when the population is equal to 1. For the dynamic instances, results for runs with each one of the 7 change types are presented. We also tested a *mixed* strategy, where the change type is randomly chosen for each environment where $\text{mod}(e, 2) = 1$. In the mixed strategy, different change types occur in a run. The strategies described in Section 6 (*MI*, *RI*, and *mInc*) are tested in algorithms applied to dynamic and static instances. Despite being proposed for dynamic instances, we tested the strategies also in static instances because we want to see if a better performance of the algorithm is in fact a result of changing the fitness landscape or not.

We want to test the hypothesis that changing the fitness landscape using the proposed framework can result in finding faster good solutions. In order to do this, the execution time for each run of the algorithms in static and dynamic instances was fixed, i.e., all algorithms employ the same computational resources. A better result for the best fitness found by the algorithm indicates a faster execution of the algorithm, i.e., a better speed up. For dynamic instances, the best result is stored only for environments where $d(e) = 0$. Thus, best results for the algorithms in dynamic and static instances can be compared. The results of 50 runs for each combination of dimension ($l$), algorithm, change severity ($\rho$), and DOP strategy are presented. Since the execution time is fixed, the number of evaluations and generations for the algorithms can be different. The Wilcoxon signed-rank test with the confidence level 0.95 is used to test the statistical significance of the results. We present the results in a compact form in the tables and figures of this paper. The corresponding results, e.g., the average fitness, can be seen in the

supplementary material of this paper.

### *7.2. Experimental Results: Pseudo-Boolean Optimization Problems*

Two pseudo-Boolean optimization problems are used to test the proposed framework and strategies. The first problem is the NK landscapes Kauffman (1993). The evaluation function for the NK landscapes is composed of $l$ sub-functions[2], each one influenced by $K+1$ elements of the candidate vector $\mathbf{x}$. The evaluation function is given by:

$$f_P(\mathbf{x}) = \sum_{i=1}^{l} f_i(\mathbf{x}, \mathbf{m}_i) \tag{4}$$

where $\mathbf{x} \in \mathbb{B}^l$ is the candidate solution and $\mathbf{m}_i \in \mathbb{B}^l$ is a mask that indicates the elements of $\mathbf{x}$ that influence sub-function $f_i$. The element of $\mathbf{m}_i$ at position $i$ and $K$ other elements are equal to one. The evaluation function of the NK landscapes is a $k$-bounded pseudo-Boolean function, where $k = K+1$. The evaluation of each sub-function for each combination of inputs is randomly generated in the range $[0,1]$. Here, the adjacent model is adopted, i.e., the elements of $\mathbf{x}$ that influence $f_i$ are adjacent, $K = 2$ and the objective is to maximize Eq. (4).

The second problem is the 0-1 knapsack problem Han and Kim (2000), with fitness function given by:

$$f_P(\mathbf{x}) = \sum_{i=1}^{l} p_i x_i - R(\mathbf{x}) \tag{5}$$

where the candidate solution $\mathbf{x} \in \mathbb{B}^l$ represents a subset of items in the knapsack. The $i$-th item, $i = 1, \ldots, l$, has weight $w_i \in \mathbb{R}^+$ and profit $p_i \in \mathbb{R}^+$. In the experiments presented here, the weights are randomly generated in the range $[5, 20]$ and the profits are randomly generated in the range $[40, 100]$. The knapsack capacity $C$ is equal to 50% of the sum of all weights. In Eq. (5), the penalty term $R(\mathbf{x})$ is given by:

$$R(\mathbf{x}) = \begin{cases} 0, & \text{if } \sum_{i=1}^{l} w_i x_i \leq C \\ \alpha\left(\sum_{i=1}^{l} w_i x_i - C\right), & \text{otherwise} \end{cases} \tag{6}$$

where $\alpha = \max_{i=1,\ldots,l}(p_i/w_i)$. The objective is to maximize Eq. (5).

[2]For uniformity in notations, we adopt $l$ for the number of sub-functions (and size of the candidate vector), instead of $N$.

Table 4: Algorithms.

| Problem | Algorithm | $l$ | Runtime (s) | $\tau$ |
|---|---|---|---|---|
| NK landscapes | (1+1)-EA | 100 | $l$ | 1000 |
| | GA | 100 | $l$ | 500 |
| 0-1 knapsack | GA | 200 | $l$ | 500 |
| $f_6$ to $f_{12}$ | (1+1)-ES | 10 | $l$ | 1000 |
| | $(\mu,\lambda)$-ES | 10 | $l$ | 300 |
| | $(\mu,\lambda)$-ES | 30 | $3l$ | 300 |

In the experiments, the evaluation of the best solution found in each run is compared to the evaluation of the global optimum. We analyze the number of times that the global optimum was found by an algorithm, considering all runs. For both problems, the global optimum is obtained by dynamic programming. The time complexity of dynamic programming for the 0-1 knapsack problem is $O(lC)$. If $C$ is polynomial, the algorithm runs in a polynomial time. Dynamic programming is also polynomial for the adjacent model of the NK landscapes. However, both problems are NP-complete for the general case.

In the experiments with the pseudo-Boolean optimization problems, all algorithms run for $l$ seconds. The change period, $\tau$, and the algorithms used for each problem are shown in Table 4. For the GA, the population size is set to 100, and the tournament selection, elitism, uniform crossover with rate 0.6, and bit flip mutation with rate $1/l$ are employed. In the tournament selection, the best in a pool of 3 individuals randomly chosen is selected. We test GAs with: i) no immigrant strategies; ii) with random immigrants strategy only; iii) with memory immigrants strategy only; iv) with both immigrant strategies. For the NK landscapes, an EA with one solution, denoted (1+1)-EA, is also employed. In the (1+1)-EA, if an offspring generated by flip mutation is better than its parent, it replaces the parent. As commented before, the (1+1)-EA is employed because we want to investigate if an eventual better performance of changing the problem is due only to an increase in the diversity of the population.

The percentage of runs where the global optimum was found by (1+1)-EA for static and dynamic instances of the NK landscapes is presented in Table 5[3]. For the static instances, different kick sizes (0, 0.1, 0.2, 0.5, 1.0) are considered. For the dynamic instances generated for each change type, different levels of change

[3]The tables in the supplementary material show the average errors (over 50 runs) for static and dynamic instances for pseudo-Boolean and continuous optimization problems. The results of the Wilcoxon signed-rank test used to test the statistical significance are also presented.

Table 5: Percentage over 50 runs where the global optimum is found by (1+1)-EA for the NK landscapes. The best result for the static instance with different kick sizes is compared to the results for dynamic instances of different types, all with $\tau = 1000$. The duration of each run for static and dynamic instances is equal ($l$ seconds). The results for the static instance correspond to the best results among the experiments with different kick sizes (0, 0.1, 0.2, 0.5, 1.0). Bold face indicates that the result for the dynamic environment is better than the respective result for the static environment. The best result is indicated by symbol $*$.

| | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| 0.001 | 6 | **46** | **42** | **14** | **48*** | 6 | 2 | **10** | **10** |
| 0.010 | | **46** | **42** | **10** | **48*** | **12** | 0 | **36** | **8** |
| 0.100 | | **8** | 4 | **10** | **24** | 6 | 0 | **44** | 6 |
| 0.200 | | 0 | 6 | 4 | **16** | 2 | 0 | **34** | 4 |
| 0.500 | | 0 | 0 | 4 | 0 | 6 | 0 | **34** | 0 |
| 0.900 | | 0 | 2 | 0 | 0 | 2 | 0 | **38** | 0 |

Table 6: Percentage over 50 runs where the global optimum is found by GA for the NK landscapes problem. When the problem changes, the change period ($\tau$) is equal to 500 generations. For each change severity $\rho$, four algorithms are tested, where random immigrants (strategy *RI*) and memory immigrants (strategy *MI*) are inserted or not after each change. The duration of each run is $l$ seconds. Bold face indicates that the result for the dynamic environment is better than the respective result for the static environment.

| | | | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 28 | 24 | 26 | **54** | **42** | **50** | 22 | **32** | **48** |
| no | no | 0.010 | | 24 | 26 | **50** | **42** | **56** | 22 | **38** | **54** |
| no | no | 0.100 | | **70** | **72** | **66** | **76*** | **64** | 22 | **50** | **66** |
| no | yes | 0.001 | 28 | 24 | **32** | **60** | **58** | **44** | 18 | **30** | **52** |
| no | yes | 0.010 | | 24 | **32** | **60** | **58** | **62** | 18 | **48** | **52** |
| no | yes | 0.100 | | 20 | 22 | **66** | **70** | **72** | 18 | **54** | **46** |
| yes | no | 0.001 | 34 | 28 | 28 | **52** | **44** | **52** | 22 | 32 | **38** |
| yes | no | 0.010 | | 28 | 28 | **60** | **44** | **62** | 22 | **42** | **50** |
| yes | no | 0.100 | | **64** | **68** | **56** | **68** | **62** | 22 | **50** | **66** |
| yes | yes | 0.001 | 24 | 18 | 18 | **54** | **54** | **56** | 18 | **30** | **44** |
| yes | yes | 0.010 | | 18 | 18 | **52** | **54** | **62** | 18 | **36** | **48** |
| yes | yes | 0.100 | | 20 | **28** | **68** | **62** | **52** | 18 | **46** | **54** |

severity ($\rho$) are considered.

Changing the environment results in better performance, but not for all change types and values of $\rho$. However, for some change types and values of $\rho$, the performance in the dynamic instances is much better. Change Type 3.1 results in better performance when dynamic instances are compared to static instances for all values of $\rho$. The best performance is reached for Change Type 2.1 with $\rho < 0.1$, where (1+1)-EA finds the global optimum in 48% of the runs, against 6% for the static instances. It is important to observe that the results shown in

Table 7: Percentage over 50 runs where the global optimum is found by GA for the 0-1 knapsack problem.

| | | | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 64 | **92** | **90** | **90** | **84** | **90** | 38 | 64 | **90** |
| no | no | 0.010 | | **86** | **88** | **90** | **84** | **90** | 38 | **72** | **86** |
| no | no | 0.100 | | **66** | **66** | **86** | **76** | **88** | 42 | **86** | **84** |
| no | yes | 0.001 | 58 | **70** | **66** | **82** | **66** | **86** | 52 | **68** | **78** |
| no | yes | 0.010 | | **68** | **70** | **80** | **66** | **76** | 32 | **66** | **72** |
| no | yes | 0.100 | | 50 | 50 | **72** | 48 | **70** | 46 | **76** | **72** |
| yes | no | 0.001 | 56 | **94*** | **90** | **94*** | **86** | **86** | 42 | **58** | **86** |
| yes | no | 0.010 | | **92** | **94*** | **86** | **86** | **92** | 34 | **78** | **86** |
| yes | no | 0.100 | | **60** | **70** | **86** | **84** | **88** | 40 | **86** | **88** |
| yes | yes | 0.001 | 62 | 58 | **76** | **82** | **74** | **78** | 40 | **74** | **72** |
| yes | yes | 0.010 | | **64** | **64** | **78** | **74** | **78** | 62 | **74** | **70** |
| yes | yes | 0.100 | | 52 | 30 | **84** | 58 | **76** | 54 | **78** | **70** |

Table 5 for the static instances are for the best results among the experiments with different kick sizes. When comparing the results for different values of $\rho$, we observe that the best results are obtained by smaller values of $\rho$. Hence, we adopt $\rho = \{0.1, 0.01, 0.001\}$ in the following experiments for the GA.

Table 6 shows the percentage of runs where the global optimum is found by the GA for static and dynamic instances of the NK landscapes. As expected, better results, when compared to (1+1)-EA, are reached. The strategies *MI* and *RI* are tested, even in the static instances in order to check if better results are due to the use of the proposed strategies or to the use of the proposed framework. Again, the best results are obtained for the dynamic instances. When *MI* and *RI* are not used, Change Type 2.1 results in finding the global optimum in 76% of the runs, against 34% that is the best result for the static instances. Changes Types 1.3, 2.2, and 3.1 also result in very good results.

Better results for the dynamic instances are also obtained in the experiments with the 0-1 knapsack problem (Table 7). The best results are obtained for Change Types 1.1, 1.2 and 1.3 when the strategy *RI* is employed. For those change types, the global optimum is found in 94% of the runs, against 64% that is the best result for the static instances. For all experiments with pseudo-Boolean optimization problems, dynamic instances generated by Change Type 2.3 result in worse performance, when compared to the static instances.

In summary, changing the environments resulted in better performance for some change types, but not for all. In particular, when the GA was employed, Change Types 1.3, 2.1, and 2.2 resulted in better performance, when compared to the static instances, for both problems, strategies, and values of $\rho$. When

Table 8: Test functions for the experiments on continuous optimization. The evaluation of the global optimum, $\mathbf{x}^*$, is given in the third column, while the range for each element of $\mathbf{x}$ is given in the fourth column. Some properties of the functions are indicated in the fifth column.

| Problem | Name | $f_P(\mathbf{x}^*)$ | Range | Properties[1] |
|---|---|---|---|---|
| $f_6$ | Shifted Rosenbrock's Function | 390 | [-100,100] | NS, NV |
| $f_7$ | Shifted Rotated Griewank's Function without Bounds | -180 | $[-\infty,\infty]$ | NS, R, OR |
| $f_8$ | Shifted Rotated Ackley's Function with Global Optimum on Bounds | -140 | [-32,32] | NS, R, B |
| $f_9$ | Shifted Rastrigin's Function | -330 | [-5,5] | S, LO |
| $f_{10}$ | Shifted Rotated Rastrigin's Function | -330 | [-5,5] | NS, R, LO |
| $f_{11}$ | Shifted Rotated Weierstrass' Function | 90 | [-0.5,0.5] | NS, R |
| $f_{12}$ | Schwefel's Problem 2.13 | -460 | $[-\pi,\pi]$ | NS |

[1] S: separable, NS: non-separable, R: rotated; B: with $\mathbf{x}^*$ on bounds, OR: with $\mathbf{x}^*$ outside of the initialization range
NV: with a very narrow valley from local optimum to global optimum, LO: with a huge number of local optima

mixed change types occur, dynamic instances also resulted in better performance. Change Type 3.1 also presents good results, while Change Types 1.1 and 1.2 present good results only for the 0-1 knapsack problem. Change Type 2.3 results in the worst performance; the dynamic instances generated by Change Type 2.3 present worse performance in all instances when compared to the static instances.

### *7.3. Experimental Results: Continuous Optimization Problems*

The seven basic multimodal benchmark functions described in Suganthan et al. (2005) are selected as the test suite for the proposed framework. The test functions, which should be minimized, are presented in Table 8. Figs. 2, 3, and 7 show the fitness for some of the test functions with $l = 2$. They have different properties (Table 8) that influence the optimization algorithms, making easier or harder the optimization process. We analyze the average error (over 50 runs) between the best fitness found by the algorithm and the evaluation of the global optimum.

Table 4 shows the runtime and the change frequency adopted for the experiments with continuous optimization. Two ESs are tested: (1+1)-ES and $(\mu,\lambda)$-ES, where the number of parents is $\mu = 20$ and the number of offspring is $\lambda = 140$. For the $(\mu,\lambda)$-ES, intermediate recombination is employed. For both algorithms, the $q$-Gaussian mutation is employed Tinós and Yang (2011). The $q$-Gaussian mutation has an interesting property: by continuously changing a real parameter $q$, the shape of the mutation distribution is smoothly modified. The $q$-Gaussian mutation reproduces the Gaussian mutation for $q = 1$, while it reproduces the Cauchy mutation for $q = 2$. Changing the mutation distribution is very interesting, especially in DOPs. Here, the ESs employ self-adaptation for adapting the parameter $q$ and the $l$ mutation parameters. The strategy *mInc* (Section 6) is test-

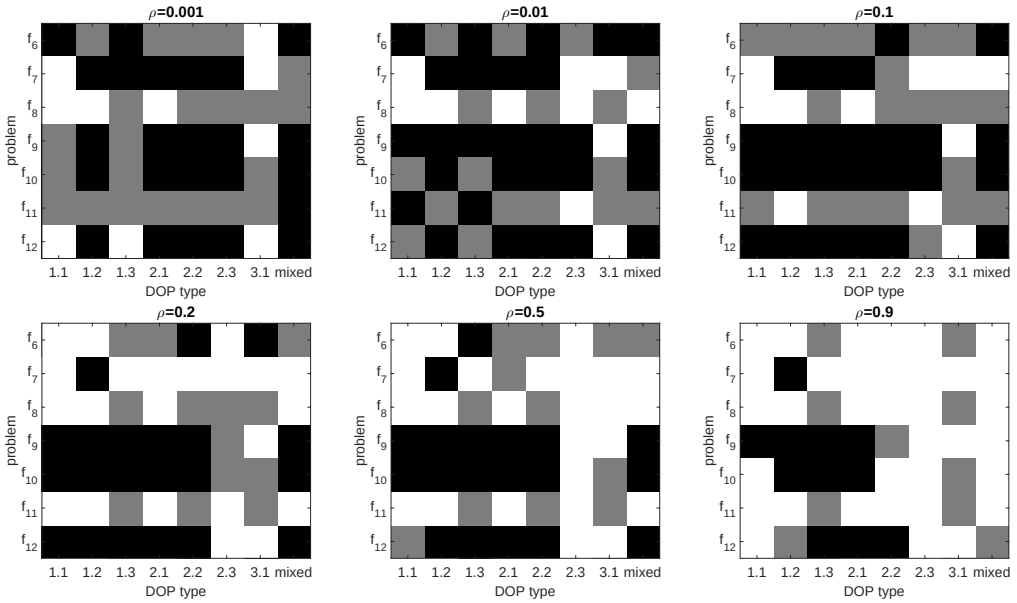


Figure 4: Performance of (1+1)-ES in the optimization of continuous functions with $l = 10$. The best result for the static instance with different kick sizes is compared to the results for dynamic instances of different types, all with $\tau = 1000$. The results for the static instance correspond to the best results among the experiments with different kick sizes (0, 0.1, 0.2, 0.5, 1.0). When the statistical test indicates no difference between the results for static and dynamic instances, a gray square is shown. Otherwise, a black or white square is shown when the result for the dynamic instance is respectively better or worse than the result for the static instance.

ed in $(\mu, \lambda)$-ES. When *mInc* is employed, $q$ is set to 2 in the generation before the change and in the generation when the change occurs. When $q = 2$ (Cauchy mutation), a higher average number of large jumps is generated when compared to the Gaussian mutation.

The performance of (1+1)-ES for static and dynamic instances of the continuous optimization problems are compared in Fig. 4[4]. Better results are obtained in dynamic instances, but not for all problems. Particularly, all the results for problem $f_8$ are worse for the dynamic instances. For $\rho \leq 0.1$, the results for the dynamic instances for Change Types 1.3 and 2.2 are better than the results for the static instances. It is important to observe that the results for the static instances correspond to the best results among the experiments with different kick sizes. For

[4]Figures 4 and 5 graphically show the comparison of the results between static and dynamic instances. Tables in the supplementary material show corresponding numerical results.

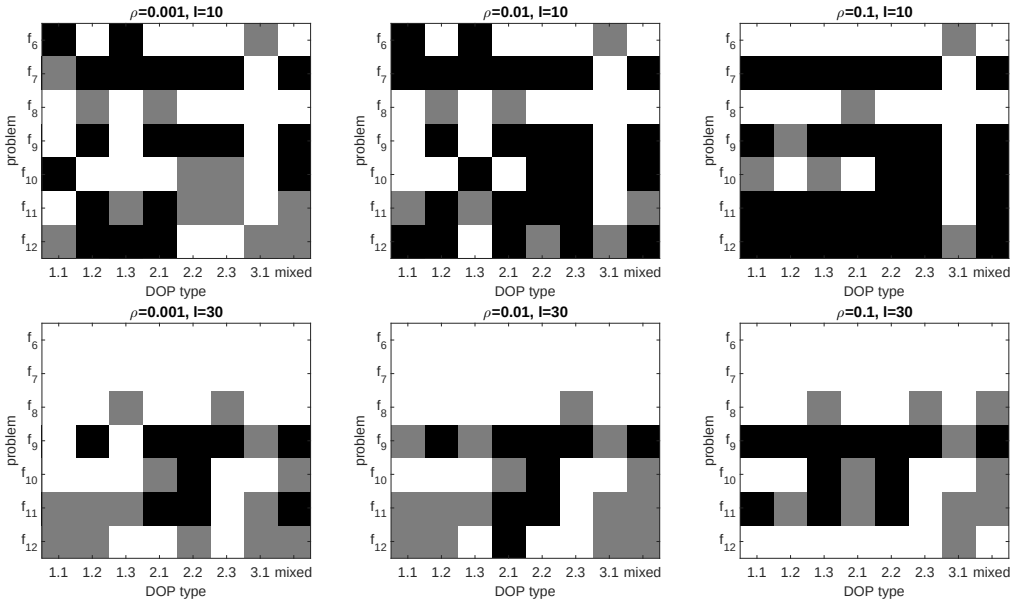


Figure 5: Performance of $(\mu,\lambda)$-ES in the optimization of continuous functions with $l = 10$ and $l = 30$. The result for the static instance is compared to the results for dynamic instances of different types, all with $\tau = 300$. When the statistical test indicates no difference between the results for static and dynamic instances, a gray square is shown. Otherwise, a black or white square is shown when the result for the dynamic instance is respectively better or worse than the result for the static instance.

$\rho = 0.1$, changing the problem results in better performance for problems $f_9$ and $f_{10}$ for all change types, with exception for Change Type 3.1. The performance for dynamic instances generated by Change Type 3.1 are worse or similar to the performance of the static instances, with exception for problem $f_6$ with $\rho = 0.2$. The best performance for the dynamic instances are generally obtained for smaller values of $\rho$. In this way, we adopt $\rho = \{0.1, 0.01, 0.001\}$ in the experiments presented in the following for $(\mu,\lambda)$-ES.

Fig. 5 shows the results for $(\mu,\lambda)$-ES in the optimization of continuous functions with $l = 10$ and $l = 30$. The *mInc* strategy was not used in these experiments. Better results for the dynamic instances are obtained when $l = 10$. For $\rho = 0.1$ and $l = 10$, changing the problem according to all change types, with exception for Change Type 3.1, resulted in better performance for problems $f_7$, $f_9$, $f_{11}$, and $f_{12}$ in 27 out of 28 times. For $l = 10$, the best results were obtained when Change Types 2.2. and 2.3 were introduced with $\rho = 0.1$; in such cases, the dynamic instances resulted in better results for 5 out of 7 problems. Worse results

are obtained when $l = 30$. However, dynamic instances for Change Type 2.2 still present better results in 3 ($f_9$, $f_{10}$, and $f_{11}$) out of 7 problems for all values of $\rho$ when $l = 30$.

The *mInc* strategy did not generally improve the performance of $(\mu, \lambda)$-ES[5]. When $\rho = 0.01$, the *mInc* strategy results in better performance for $f_9$ and for 2 out of 8 change types for $f_{10}$. The performance is worse for $f_7$, for 5 out of 8 change types for $f_6$, and for 2 out of 8 change types for $f_{10}$.

For the experiments with continuous optimization problems, changing the environments resulted in better performance for some change types and some problems, but not for all. In particular, when $(\mu, \lambda)$-ES was employed, Change Types 2.2 and 2.3 with $\rho = 0.1$ resulted in better performance for $l = 10$, when compared to the static instances. The mixed change types strategy also resulted in better performance. Change Type 3.1 results in the worst performance.

For $l = 30$, Change Type 2.2 resulted in better performance of $(\mu, \lambda)$-ES for three functions. Two of them ($f_9$ and $f_{10}$) are functions with a huge number of local optima (Table 8). Change Type 2.2 also resulted in good performance for these functions when $l = 10$ and also when (1+1)-ES was employed. Besides, all dynamic instances with Change Type 2.2 presented better performance than the corresponding static instances for the NK-landscapes (Table 6) and 0-1 knapsack (Table 7) problems. Both pseudo-Boolean problems are highly multimodal.

Changing the environment results in better performance even when the population of the EA and ES is equal to one. This result indicates that eventually increasing the diversity of the population after the changes cannot be pointed as the main explanation for improving the optimization for the proposed framework.

One can observe that the best results for the dynamic instances generally occur for small values of $\rho$. When $\rho$ is smaller, the search space fraction affected by the changes is smaller for Change Types 1.3, 2.2, 2.3, and 3.1. For Change Types 1.1 and 1.2, smaller $\rho$ implies smaller rotation for the environment. Finally, smaller $\rho$ implies smaller transformation of the search space for Change Type 2.1.

For the continuous optimization problems, we used all the basic multimodal functions employed in the 2005 IEEE Congress on Evolutionary Computation (IEEE CEC) Competition on Real Parameter Optimization Suganthan et al. (2005). There, it was suggested to run the algorithms for $10,000 \times l$ fitness evaluations. Here, we use a similar criterion: the time of each run is fixed to $3 \times l$ seconds.

[5]Figure S.1 in the supplementary material shows the results for the comparison of $(\mu, \lambda)$-ES with and without the *mInc* strategy in the optimization of continuous functions with $l = 30$.

We used the runtime instead of the number of evaluations because changing the problem affects the runtime of the algorithm but not the number of evaluations in each run. The runtime complexity of algorithms employed in the 2005 IEEE CEC Competition on Real Parameter Optimization (and also of the ES employed here) is not $O(l)$ for the basic multimodal functions used in the competition. However, the change in the number of evaluations (or the runtime) used in the competition is $O(l)$. In this way, the performance of the algorithms deteriorates when the number of dimensions increased. Here, the static instances resulted in a better performance for more instances with $l = 30$. When the average errors are analyzed[6], we can observe that they are high, indicating that the ES did not have enough time for optimizing the instances. If we run the instances for more time, local optima with better quality are found. Then, mechanisms for escaping from local optima, e.g., introducing artificial changes, generally have positive impact on the performance of the EA. This is also true for the pseudo-Boolean optimization problems[7].

It is interesting to observe that the dynamic instances generated by the proposed framework generally result in better performance for optimizing the candidates solutions for the problems investigated here. In dynamic instances, the algorithm is directly optimizing the static fitness landscape approximately 50% of the time, when compared to the respective algorithms in the static instances. Besides, because the runtime is fixed, changing the environments impacts the number of generations performed by the algorithm[8].

Figs. 3, 6, 7, 8 help in understanding why changing the environment can be useful. Figs. 6-8 show the trajectories of the solutions when (1+1)-ES is employed in problems with $l = 2$. We highlight two reasons that help explaining why changing the environment using the proposed framework can be beneficial. They are explained in the following.

*7.3.1. Generation of Plateaus in the Region Occupied by the Best Solution*

In the plateaus, the effect of selection based on fitness is neutral, i.e., two solutions in the plateau have the same probability of being selected. We can observe in

[6] See tables in the supplementary material.

[7] Results for experiments with the 0-1 knapsack problem with $l = 100$ and $l = 300$ are presented in the supplementary material. The performance of the GA for the static instances deteriorates when the dimension of the problems increases. However, dynamic instances still resulted in better performance.

[8] Figure S.2 in the supplementary material shows the number of generations in an experiment with static and dynamic instances of the 0-1 knapsack problem.

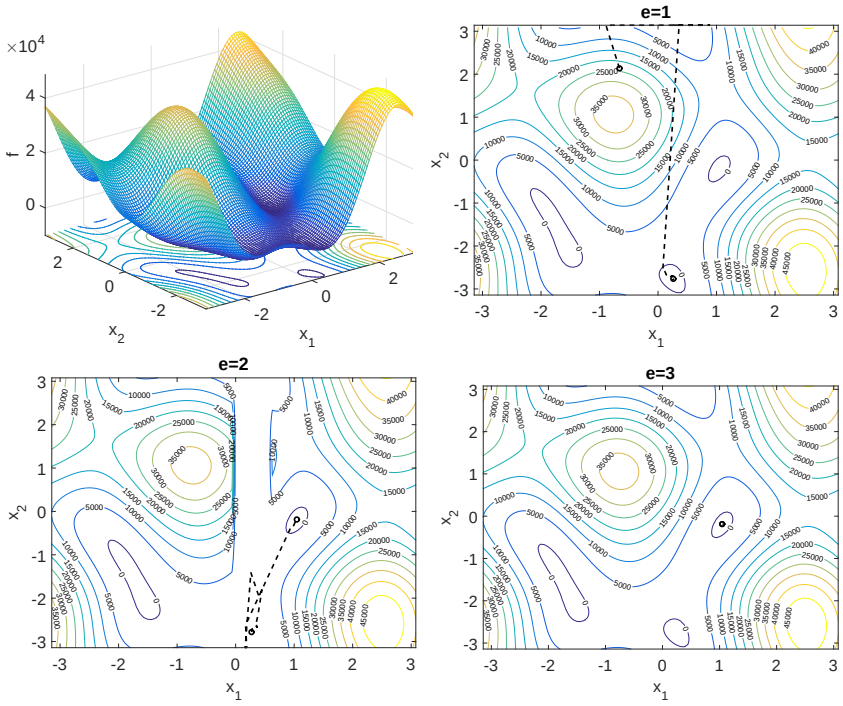


Figure 6: Trajectories for the first run in the experiment with problem $f_{12}$ with $l = 2$. Each graph shows the contour curves for the first three environments created by Change Type 2.2 with $\rho = 0.5$. The graphs for $e = 1$ and $e = 3$ correspond to the static environment (the fitness for the static environment is shown on top left).

Fig. 6 that, as a consequence, regions in the borders of the plateaus can be easily reached. Of course, the landscape of the regions close to the plateaus will have a great influence on the new dynamics of the population. If the best solution before the change, $\mathbf{b}(e-1)$, is a local optimum, the solution can escape depending on the landscape of the regions close to the plateaus. Plateaus are formed when Change Types 2.2 and 2.3 occur. According to the definition of the changes (Section 3), the plateaus are created in a region occupied by $\mathbf{b}(e-1)$.

For Change Type 2.2, the height of the plateau is equal to the fitness of a random solution inside the plateau (Fig. 3). For Change Type 2.3, the height of the plateau is equal to the fitness of solution $\mathbf{b}(e-1)$ (Fig. 3). When Change Types 2.2 and 2.3 occur, the solution is allowed to explore the plateau and the regions around it. If there is a promising region around the plateau, i.e., a region with lower fitness, the solution can move to this region. This behavior can be observed in Figs. 6 and 7. In environment $e = 2$ in Fig. 6, which was generated by Change

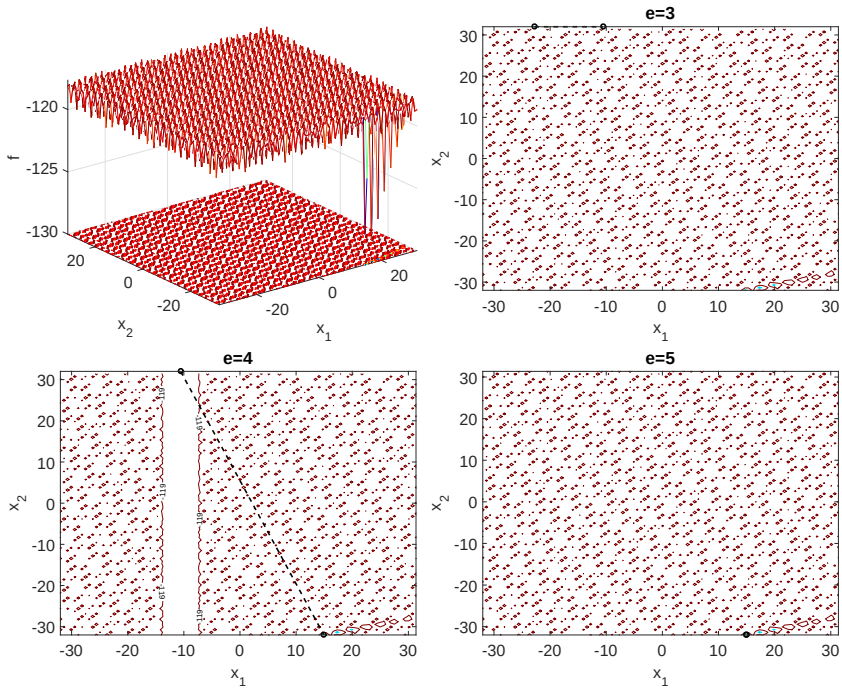


Figure 7: Initial and final solutions for three consecutive environments $e$ in the first run of the experiment with problem $f_8$ with $l = 2$. Each graph shows the contour curves for the environments created by Change Type 2.3 with $\rho = 0.5$. The graphs for $e = 3$ and $e = 5$ correspond to the static environment.

Type 2.2, the solution moved fast from a local optimum found in environment $e = 1$ to a new local optimum with better evaluation. When $e = 2$, the contour curves in Fig. 6 show that the height of the plateau is higher than the fitness around the new optimum.

For Change Type 2.3, the transition to a new local optimum also occurs, but the transition takes a longer time. This occurs because the height of the plateau is low, with the same evaluation of $\mathbf{b}(e-1)$. In this way, it is harder for the solution to move to a promising area. This helps explaining why Change Type 2.2 generally yields better results than Change Type 2.3 in the experiments. However, as the selection is neutral inside the plateau, the solution is free to move inside it and eventually to jump to a new promising area for both change types. This behavior occurred in the experiment that resulted in Fig. 7. In this figure, we do not show the whole trajectory for environment $e = 4$ because the solution moved around all the plateau and the resulting figure would have too many points. In this way, we show only the initial and final points of the trajectory for each environment.

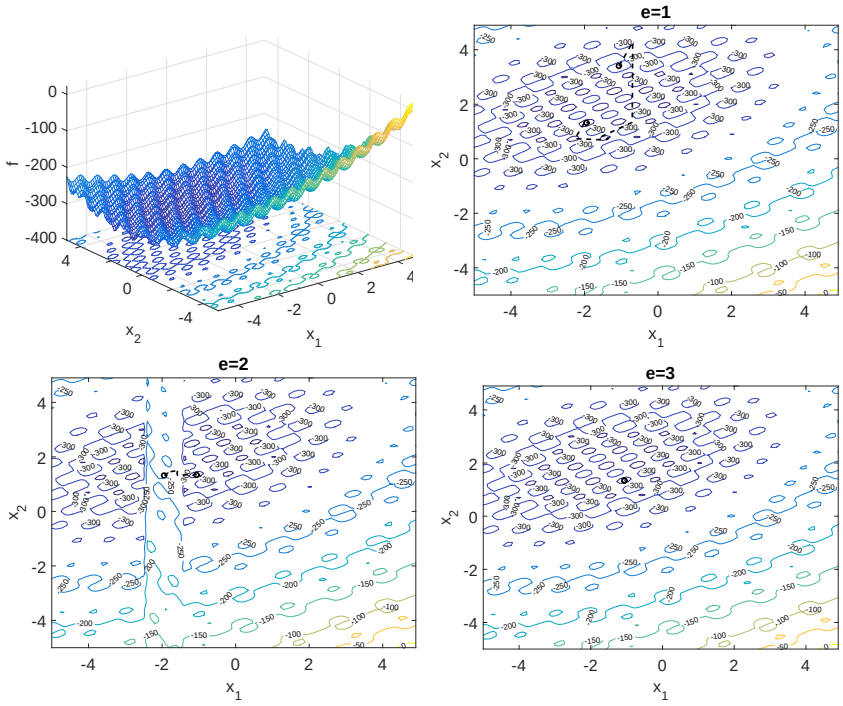


Figure 8: Trajectories for the first run of the experiment with problem $f_{10}$ with $l = 2$. Each graph shows the contour curves for the first three environments created by Change Type 1.3 with $\rho = 0.5$. The graphs for $e = 1$ and $e = 3$ correspond to the static environment.

Moving to a new area around the plateau does not always happen and many times the solution moves back to the old local optimum when the environment changes again. However, eventually a new promising area is reached, allowing the solution to escape from a local optimum valley. Creating a plateau around the found best solution is similar to creating a tabu list during $\tau$ generations, which avoids returning to the local optimum. Creating plateaus explains the good performance of Change Types 2.2 and 2.3 in the continuous optimization problems. Dynamic instances generated by Change Types 2.2 and 2.3 produced better performance than the static instances for all functions with $l = 10$, except for functions $f_6$ and $f_8$. Function $f_8$ (Fig. 7) is very difficult for the ESs; in this case, the runtime for $l = 10$ and $l = 30$ is not enough for producing local optima close to the global optima. However, for the remaining functions, introducing plateaus allows the algorithm to reach better solutions faster, especially for the functions with a huge number of local optima ($f_9$ and $f_{10}$).

*7.3.2. Modification of the Gradient of the Fitness Landscape in the Region Occupied by the Best Solution*

Another potential benefit of the changes is modifying the gradient of the region around $\mathbf{b}(e-1)$. This can be observed in Fig. 2 for Change Types 1.1, 1.2, 1.3, 2.1 and 3.1. While the landscape is modified by rotation for Change Types 1.1, 1.2, and 1.3, it is caused by adding a deviation for the subset of solutions affected by the change. When comparing the changes that affect almost all the search space (Change Types 1.1, 1.2, 2.1) with the changes that affect only a small subset (Change Types 1.3, 2.2, 2.3, 3.1), we can see that changing only a subset can be beneficial for finding the optimum for the static space faster because they preserve most of the landscape. Those changes affect only a region close to the current best solution, allowing the exploration of other parts of the search space. The effects of the changes in modifying the landscapes, particularly the basis of attraction around local optima, can be observed in the contour curves in Fig. 2.

An illustrative example of the effect of the modification of the gradient of the region occupied by the current best individual can be observed in Fig. 8. The fitness landscape of the problem illustrated in the figure has a huge number of local optima. When Change Type 1.3 occurs, the gradient of the region around the best solution before the change is modified. This modification allows the solution to escape from the local optimum and reach a better optimum. The modification of the gradient helps explaining the good results for Change Types 1.1, 1.2, 1.3, 2.1 and 3.1. In the experiments, Change Types 1.3, 2.1, and 3.1 presented very good results in the pseudo-Boolean problems. Change Types 1.3 and 2.1 also presented good results in the continuous problems. In particular for function $f_6$, Change Types 1.1 and 1.3 produced better results in the experiments with $l = 10$ for $\rho \leq 0.01$. Functions $f_6$ and $f_8$ are the only two functions where Change Types 2.2 and 2.3 did not produced better results for $\rho = 0.1$ than the results for the static instances. Function $f_6$ has few local optima and has a very narrow valley from local optima to the global optima Suganthan et al. (2005). Thus, generating plateaus (by using Change Types 2.2 and 2.3) did not result in better performance. However, changing the gradient of the fitness landscape around the region occupied by the current best solution can result in a better performance.

## 8. Conclusions

We investigated in this paper the impact of artificial changes on the optimization speed up. A framework for inducing artificial changes is proposed[9]. Given a pseudo-Boolean or continuous optimization static problem, the proposed framework can be used to change dynamically the optimization problem in order to improve the optimization speed up. The change types can be controlled, as well the change severity.

Seven different types of changes can be induced. They have different properties, changing the dynamics of the population in different ways. For example, Change Types 2.2 and 2.3 create plateaus in a small region occupied by the best solution found in the environment before the change. The other change types modify the gradient of the fitness landscape around the current solutions. For example, Change Types 1.1, 1.2, and 1.3 modify the gradient of the fitness landscape by rotating parts of the search space or all of it. Change Type 2.1 changes the gradient by copying some decision variables, while Change Type 3.1 does so by increasing or decreasing the fitness in a small region occupied by the best solution found in the environment before the change. Introducing plateaus or changing the gradient of regions of the fitness landscape can help the population to escape from local optima. As a consequence, the optimization process can be speeded up.

Experiments with two pseudo-Boolean problems (NK landscapes and 0-1 knapsack problem) and seven continuous optimization problems show that better performance is obtained for dynamic instances when compared to static instances, even when an iterated local search strategy is considered. This better performance is not for all change types and change severity values. In fact, some change types did not produce good performance, particularly Change Type 2.3 in the pseudo-Boolean optimization problems and Change Type 3.1 in the continuous optimization problems.

However, some change types produced very good results. For the NK landscapes, the global optimum was found in 76% of the runs, against 34% for static instances. For the 0-1 knapsack problem, this result was 94% of the runs for dynamic instances, against 64% for static instances. For the continuous optimization problems, better performance was also obtained for dynamic instances, however, not for all problems. Depending on the properties of a problem, better or worse performance is obtained. Particularly, Change Type 2.2 resulted in better performance for continuous problems with huge number of local optima. Change Type

[9]The source code for the proposed framework is available at https://github.com/rtinos/iDOP.

2.2 resulted also in better performance for the NK landscapes and 0-1 knapsack problem; both are highly multimodal. In this way, changing the fitness landscape for some change types using the proposed framework represents an additional approach when dealing with highly multimodal problems. It is important to observe that the proposed methodology can be used with other approaches, e.g., increasing mutation rates or restarting populations.

Some of the best results for the dynamic instances were obtained by three simple strategies proposed in this paper. The proposed strategies use information about the changes, e.g., time of occurrence of the change. Such information is not available in intrinsic DOPs, where we do not know when and how the change occurs.

The investigation of artificially induced DOPs opens new research possibilities in EDO. Several future works are possible. It is necessary to investigate new strategies and operators that make use of the knowledge about the changes. It is still necessary to theoretically investigate how and when to change the environment. The changes should be induced according to the objectives of the programmer, which makes the theoretical investigation harder. Concerning the proposed framework, new change types can be investigated. The framework can also be adapted for generating continuous intrinsic benchmark DOPs.

## Acknowledgement

This work was supported by FAPESP and CNPq, Brazil, under grants 2013/07375-0, 2015/06462-1, 2016/18615-0, and 304400/2014-9, and the Engineering and Physical Sciences Research Council (EPSRC) of the U.K. under Grant EP/K001310/1.

## References

Branke, J., 2001. Evolutionary Optimization in Dynamic Environments. Kluwer.

Cobb, H. G., Grefenstette, J. J., 1993. Genetic algorithms for tracking changing environments. In: Proc. of the 5th Int. Conf. on Genetic Alg. pp. 523–530.

Cruz, C., González, J., Pelta, D., 2011. Optimization in dynamic environments: a survey on problems, methods and measures. Soft Computing 15, 1427–1448.

Eiben, A. E., Smith, J., 2015. From evolutionary computation to the evolution of things. Nature 521 (7553), 476–482.

Eriksson, R., Olsson, B., 2004. On the performance of evolutionary algorithms with life-time adaptation in dynamic fitness landscapes. In: Proc. of the 2004 IEEE Cong. on Evolutionary Computation. Vol. 2. pp. 1293–1300.

Eyckelhof, C. J., Snoek, M., Vof, M., 2002. Ant systems for a dynamic TSP: Ants caught in a traffic jam. In: Ant Algorithms : Third International Workshop (LNCC2463). Springer Verlag, pp. 88–99.

Fu, H., Sendhoff, B., Tang, K., Yao, X., 2015. Robust optimization over time: Problem difficulties and benchmark problems. IEEE Trans. on Evol. Comp. 19 (5), 731–745.

Han, K.-H., Kim, J.-H., 2000. Genetic quantum algorithm and its application to combinatorial optimization problem. In: Proc. of the IEEE Cong. on Evol. Comp. Vol. 2. pp. 1354–1360.

Helbig, M., Engelbrecht, A. P., 2014. Benchmarks for dynamic multi-objective optimisation algorithms. ACM Computing Surveys (CSUR) 46 (3), 37.

Kashtan, N., Noor, E., Alon, U., 2007. Varying environments can speed up evolution. Proc. of the National Academy of Sciences 104 (34), 13711–13716.

Kauffman, S. A., 1993. The origins of order: Self-organization and selection in evolution. Oxford university press.

Knowles, J. D., Watson, R. A., Corne, D. W., 2001. Reducing local optima in single-objective problems by multi-objectivization. In: Evolutionary multi-criterion optimization. Springer, pp. 269–283.

Li, C., Yang, S., 2008. A generalized approach to construct benchmark problems for dynamic optimization. In: Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K., Branke, J., Shi, Y. (Eds.), Simulated Evolution and Learning. Vol. 5361 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 391–400.

Mavrovouniotis, M., Li, C., Yang, S., 2017. A survey of swarm intelligence for dynamic optimization: Algorithms and applications. Swarm and Evolutionary Computation 33, 1–17.

Mavrovouniotis, M., Yang, S., 2015. Ant algorithms with immigrants schemes for the dynamic vehicle routing problem. Information Sciences 294, 456–477.

Mavrovouniotis, M., Yang, S., Yao, ., 2012. A benchmark generator for dynamic permutation-encoded problems. In: International Conference on Parallel Problem Solving from Nature. Springer, pp. 508–517.

Morrison, R. W., 2004. Designing evolutionary algorithms for dynamic environments. Springer-Verlag New York Inc.

Nguyen, T. T., Yang, S., Branke, J., 2012. Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evol. Comp. 6, 1–24.

Nguyen, T. T., Yao, X., 2012. Continuous dynamic constrained optimizationthe challenges. IEEE Transactions on Evolutionary Computation 16 (6), 769–786.

Otwinowski, J., Tanase-Nicola, S., Nemenman, I., 2011. Speeding up evolutionary search by small fitness fluctuations. Journal of Statistical Physics 144 (2), 367.

Parter, M., Kashtan, N., Alon, U., 2008. Facilitated variation: how evolution learns from past environments to generalize to new environments. PLOS Comput. Biology 4 (11), e1000206.

Richter, H., 2015. Coevolutionary intransitivity in games: A landscape analysis. In: Applications of Evol. Comp. Springer, pp. 869–881.

Rohlfshagen, P., Yao, X., 2009. The dynamic knapsack problem revisited: A new benchmark problem for dynamic combinatorial optimisation. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G., Ekrt, A., Esparcia-Alczar, A., Farooq, M., Fink, A., Machado, P. (Eds.), Applications of Evolutionary Computing. Vol. 5484 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 745–754.

Steinberg, B., Ostermeier, M., 2016. Environmental changes bridge evolutionary valleys. Science Advances 2 (1), e1500921.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., Tiwari, S., 2005. Problem definitions and evaluation criteria for the cec 2005 special session on real parameter optimization. Tech. rep., Nanyang Technological University.

Tan, L., Gore, J., 2012. Slowly switching between environments facilitates reverse evolution in small populations. Evolution 66 (10), 3144–3154.

Tinós, R., Yang, S., 2007. Continuous dynamic problem generators for evolutionary algorithms. In: 2007 IEEE Congress on Evolutionary Computation. pp. 236–243.

Tinós, R., Yang, S., 2011. Use of the q-gaussian mutation in evolutionary algorithms. Soft Computing 15 (8), 1523–1549.

Tinós, R., Yang, S., 2014. Analysis of fitness landscape modifications in evolutionary dynamic optimization. Information Sciences 282, 214–236.

Tinós, R., Yang, S., 2016. Artificially inducing environmental changes in evolutionary dynamic optimization. In: Handl, J., Hart, E., Lewis, P., López-Ibáñez, M., Ochoa, G., Paechter, B. (Eds.), Parallel Problem Solving from Nature PPSN XIV. Lecture Notes in Computer Science. Vol. 9921. pp. 225–236.

Yang, S., 2003. Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proc. of the 2003 Cong. on Evolutionary Computation. Vol. 3. pp. 2246–2253.

Yang, S., 2008. Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evol. Comp. 16 (3), 385–416.

Younes, A., Basir, O., Calamai, P., 2003. A benchmark generator for dynamic optimization. In: Proc. of the 3rd Int. Conf. on Soft Computing, Opt., Simulation and Manufacturing Syst.

Zhou, R., Nee, A. Y. C., Lee, H. P., 2009. Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems. International Journal of Production Research 47 (11), 2903–2920.

# A Framework for Inducing Artificial Changes in Optimization Problems —Supplementary Material

Renato Tinós and Shengxiang Yang

This is the supplementary material to the paper entitled "A Framework for Inducing Artificial Changes in Optimization Problems", published in Information Sciences. This material provides: i) the average error over 50 runs of algorithms for static and dynamic instances; ii) results for experiments with the 0-1 knapsack problem with $l = 100$ and $l = 300$ dimensions; iii) comparison of $(\mu, \lambda)$-ES with and without $mInc$ strategy in the optimization of continuous functions with $l = 30$; iv) results for an experiment comparing the number of generations for static and dynamic instances of the 0-1 knapsack problem.

Table S I: Average error (over 50 runs) for (1+1)-EA in static and dynamic instances of the NK landscapes problem. The best result for the static instance with different kick sizes is compared to the results for dynamic instances of different types, all with $\tau = 1000$. The duration of each run for static and dynamic instances is equal ($l$ seconds). The results for the static instance correspond to the best results among the experiments with different kick sizes (0, 0.1, 0.2, 0.5, 1.0). The symbol $s$ indicates that the results for the (best) static and dynamic instances are statistically different according to the Wilcoxon signed-rank test.

| | | DOP Type | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| 0.001 | 0.0016±0.0012 | 0.0007±0.0012(s) | 0.0007±0.0011(s) | 0.0014±0.0014 | 0.0005±0.0008(s) | 0.0017±0.0017 | 0.0054±0.0028(s) | 0.0059±0.0052(s) | 0.0014±0.0015 |
| 0.010 | | 0.0007±0.0012(s) | 0.0007±0.0011(s) | 0.0015±0.0015 | 0.0005±0.0008(s) | 0.0015±0.0014 | 0.0058±0.0030(s) | 0.0014±0.0017 | 0.0020±0.0020 |
| 0.100 | | 0.0016±0.0014 | 0.0018±0.0015 | 0.0018±0.0017 | 0.0010±0.0011(s) | 0.0024±0.0017(s) | 0.0059±0.0031(s) | 0.0007±0.0010(s) | 0.0021±0.0018 |
| 0.200 | | 0.0028±0.0020(s) | 0.0024±0.0018(s) | 0.0022±0.0018(s) | 0.0014±0.0015 | 0.0025±0.0021(s) | 0.0064±0.0028(s) | 0.0007±0.0010(s) | 0.0024±0.0018(s) |
| 0.500 | | 0.0053±0.0028(s) | 0.0039±0.0020(s) | 0.0021±0.0017(s) | 0.0025±0.0018(s) | 0.0031±0.0023(s) | 0.0064±0.0027(s) | 0.0008±0.0011(s) | 0.0033±0.0022(s) |
| 0.900 | | 0.0085±0.0030(s) | 0.0047±0.0023(s) | 0.0033±0.0022(s) | 0.0035±0.0022(s) | 0.0035±0.0025(s) | 0.0065±0.0026(s) | 0.0009±0.0013(s) | 0.0042±0.0026(s) |

Table S II: Average error (over 50 runs) for GA in static and dynamic instances of the NK landscapes problem. The result for the static instance is compared to the results for dynamic instances of different types, all with $\tau = 500$. The duration of each run for static and dynamic instances is equal ($l$ seconds).

| | | | | DOP Type | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 0.0020±0.0026 | 0.0019±0.0022 | 0.0018±0.0021 | 0.0008±0.0012(s) | 0.0010±0.0016(s) | 0.0008±0.0012(s) | 0.0024±0.0027 | 0.0016±0.0023 | 0.0008±0.0015(s) |
| no | no | 0.010 | | 0.0019±0.0022 | 0.0018±0.0021 | 0.0009±0.0015(s) | 0.0010±0.0016(s) | 0.0008±0.0014(s) | 0.0023±0.0027 | 0.0011±0.0013(s) | 0.0008±0.0012(s) |
| no | no | 0.100 | | 0.0002±0.0004(s) | 0.0003±0.0007(s) | 0.0006±0.0013(s) | 0.0003±0.0009(s) | 0.0006±0.0014(s) | 0.0023±0.0027 | 0.0008±0.0014(s) | 0.0003±0.0007(s) |
| no | yes | 0.001 | 0.0017±0.0023 | 0.0019±0.0024 | 0.0017±0.0023 | 0.0006±0.0011(s) | 0.0006±0.0012(s) | 0.0008±0.0011(s) | 0.0026±0.0028(s) | 0.0019±0.0024 | 0.0010±0.0017 |
| no | yes | 0.010 | | 0.0019±0.0024 | 0.0017±0.0023 | 0.0007±0.0013(s) | 0.0006±0.0012(s) | 0.0007±0.0013(s) | 0.0026±0.0028(s) | 0.0012±0.0018(s) | 0.0009±0.0013(s) |
| no | yes | 0.100 | | 0.0026±0.0029(s) | 0.0021±0.0024 | 0.0006±0.0011(s) | 0.0005±0.0010(s) | 0.0003±0.0006(s) | 0.0026±0.0028(s) | 0.0008±0.0016(s) | 0.0011±0.0018(s) |
| yes | no | 0.001 | 0.0018±0.0026 | 0.0017±0.0023 | 0.0017±0.0019 | 0.0008±0.0012(s) | 0.0009±0.0013(s) | 0.0007±0.0012(s) | 0.0023±0.0027(s) | 0.0017±0.0025 | 0.0016±0.0022 |
| yes | no | 0.010 | | 0.0017±0.0023 | 0.0017±0.0019 | 0.0006±0.0011(s) | 0.0009±0.0013(s) | 0.0008±0.0015(s) | 0.0022±0.0026 | 0.0010±0.0014(s) | 0.0011±0.0019(s) |
| yes | no | 0.100 | | 0.0006±0.0011(s) | 0.0003±0.0008(s) | 0.0007±0.0011(s) | 0.0004±0.0008(s) | 0.0007±0.0015(s) | 0.0024±0.0027(s) | 0.0009±0.0017(s) | 0.0003±0.0007(s) |
| yes | yes | 0.001 | 0.0018±0.0020 | 0.0023±0.0027 | 0.0025±0.0028 | 0.0007±0.0010(s) | 0.0008±0.0013(s) | 0.0006±0.0011(s) | 0.0024±0.0026 | 0.0021±0.0027 | 0.0013±0.0020(s) |
| yes | yes | 0.010 | | 0.0023±0.0027 | 0.0025±0.0028 | 0.0011±0.0020(s) | 0.0008±0.0013(s) | 0.0006±0.0012(s) | 0.0023±0.0026 | 0.0013±0.0018 | 0.0010±0.0020(s) |
| yes | yes | 0.100 | | 0.0023±0.0025 | 0.0021±0.0026 | 0.0003±0.0007(s) | 0.0004±0.0009(s) | 0.0009±0.0017(s) | 0.0025±0.0028 | 0.0011±0.0019(s) | 0.0009±0.0018(s) |

Table S III: Average error (over 50 runs) for GA in static and dynamic instances of the 0-1 knapsack problem with $l = 200$. The result for the static instance is compared to the results for dynamic instances of different types, all with $\tau = 500$. The duration of each run for static and dynamic instances is equal ($l$ seconds).

| | | | | DOP Type | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 0.7400±1.3063 | 0.0800±0.2740(s) | 0.1000±0.3030(s) | 0.1400±0.4522(s) | 0.2000±0.4949(s) | 0.1000±0.3030(s) | 1.6000±1.7379(s) | 0.7800±1.4039 | 0.1800±0.5956(s) |
| no | no | 0.010 | | 0.1600±0.4219(s) | 0.1200±0.3283(s) | 0.1200±0.3854(s) | 0.2000±0.4949(s) | 0.1400±0.4522(s) | 1.8200±2.0070(s) | 0.5400±1.1817 | 0.1800±0.4819(s) |
| no | no | 0.100 | | 0.6200±1.0079 | 0.5800±0.9495 | 0.1600±0.4219(s) | 0.3200±0.6207 | 0.1400±0.4046(s) | 1.6600±2.0265(s) | 0.1600±0.4219(s) | 0.2200±0.5455(s) |
| no | yes | 0.001 | 0.9000±1.2976 | 0.6200±1.1761 | 0.6800±1.1683 | 0.2000±0.4518(s) | 0.5400±0.9304 | 0.2400±0.7160(s) | 1.0600±1.4626 | 0.6400±1.0835 | 0.3600±0.7762(s) |
| no | yes | 0.010 | | 0.6000±1.0880 | 0.5400±1.0343 | 0.2400±0.5175(s) | 0.5400±0.9304 | 0.3800±0.7796(s) | 1.6000±1.7261(s) | 0.6400±1.1386 | 0.4000±0.7284(s) |
| no | yes | 0.100 | | 1.3000±1.6690 | 1.4400±1.9395 | 0.4800±0.9089(s) | 1.1600±1.5434 | 0.4200±0.7848(s) | 1.2000±1.4708 | 0.4600±0.9082(s) | 0.5600±1.0721 |
| yes | no | 0.001 | 0.9200±1.3377 | 0.0600±0.2399(s) | 0.1000±0.3030(s) | 0.0600±0.2399(s) | 0.1400±0.3505(s) | 0.1400±0.3505(s) | 1.6200±1.9680(s) | 1.1200±1.6243 | 0.1600±0.4219(s) |
| yes | no | 0.010 | | 0.1000±0.3642(s) | 0.0600±0.2399(s) | 0.1600±0.4219(s) | 0.1400±0.3505(s) | 0.0800±0.2740(s) | 1.7600±1.7677(s) | 0.3800±0.8545(s) | 0.1600±0.4219(s) |
| yes | no | 0.100 | | 0.5400±0.7879 | 0.5000±1.0152 | 0.1600±0.4219(s) | 0.1600±0.3703(s) | 0.1400±0.4046(s) | 1.4800±1.6932(s) | 0.1600±0.4219(s) | 0.1200±0.3283(s) |
| yes | yes | 0.001 | 0.8400±1.4194 | 0.9400±1.4201 | 0.4600±0.9733 | 0.3600±1.2081(s) | 0.5400±1.2651 | 0.3200±0.7407(s) | 1.4400±1.6557(s) | 0.4400±0.8843 | 0.3600±0.6627(s) |
| yes | yes | 0.010 | | 0.7000±1.1650 | 0.6600±1.1178 | 0.3400±0.7453(s) | 0.5400±1.2651 | 0.2800±0.5729(s) | 0.7800±1.2824 | 0.4800±0.9089 | 0.4800±0.8862 |
| yes | yes | 0.100 | | 1.1600±1.6458 | 1.6200±1.5238(s) | 0.3200±0.8676(s) | 1.0200±1.5583 | 0.4000±0.8571 | 1.0400±1.5113 | 0.3400±0.7982(s) | 0.5400±0.9733 |

R. Tinós is with the Department of Computing and Mathematics, University of São Paulo, Ribeirão Preto, SP, 14040-110 Brazil, e-mail: rtinos@ffclrp.usp.br.
S. Yang is with the Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, U. K., e-mail: syang@dmu.ac.uk.

Table S IV: Percentage over 50 runs where the global optimum is found by GA for static and dynamic instances of the 0-1 knapsack problem with $l = 100$.

| | | | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 96 | 100 | 100 | 100 | 100 | 100 | 98 | 94 | 100 |
| no | no | 0.010 | | 100 | 100 | 100 | 100 | 100 | 92 | 100 | 100 |
| no | no | 0.100 | | 100 | 100 | 100 | 100 | 100 | 98 | 100 | 100 |
| no | yes | 0.001 | 96 | 98 | 100 | 100 | 98 | 100 | 94 | 100 | 100 |
| no | yes | 0.010 | | 98 | 100 | 100 | 98 | 100 | 98 | 100 | 100 |
| no | yes | 0.100 | | 94 | 92 | 100 | 98 | 100 | 96 | 100 | 98 |
| yes | no | 0.001 | 98 | 100 | 100 | 100 | 100 | 100 | 96 | 100 | 100 |
| yes | no | 0.010 | | 100 | 100 | 100 | 100 | 100 | 94 | 100 | 100 |
| yes | no | 0.100 | | 100 | 100 | 100 | 100 | 100 | 96 | 100 | 100 |
| yes | yes | 0.001 | 96 | 100 | 98 | 100 | 100 | 100 | 96 | 98 | 100 |
| yes | yes | 0.010 | | 100 | 98 | 100 | 100 | 100 | 100 | 98 | 100 |
| yes | yes | 0.100 | | 100 | 98 | 100 | 100 | 100 | 92 | 100 | 94 |

Table S V: Average error (over 50 runs) for GA in static and dynamic instances of the 0-1 knapsack problem with $l = 100$. The result for the static instance is compared to the results for dynamic instances of different types, all with $\tau = 500$. The duration of each run for static and dynamic instances is equal ($l$ seconds).

| | | | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 0.0400±0.1979 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0200±0.1414 | 0.1400±0.6392 | 0.0000±0.0000 |
| no | no | 0.010 | | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.1400±0.4953 | 0.0000±0.0000 | 0.0000±0.0000 |
| no | no | 0.100 | | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0400±0.2828 | 0.0000±0.0000 | 0.0000±0.0000 |
| no | yes | 0.001 | 0.0400±0.1979 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0800±0.3405 | 0.0000±0.0000 | 0.0000±0.0000 |
| no | yes | 0.010 | | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 |
| no | yes | 0.100 | | 0.0600±0.2399 | 0.1200±0.4798 | 0.0000±0.0000 | 0.0400±0.2828 | 0.0000±0.0000 | 0.0600±0.3136 | 0.0000±0.0000 | 0.0200±0.1414 |
| yes | no | 0.001 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0800±0.4445 | 0.0000±0.0000 | 0.0000±0.0000 |
| yes | no | 0.010 | | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.1000±0.4629 | 0.0000±0.0000 | 0.0000±0.0000 |
| yes | no | 0.100 | | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0400±0.1979 | 0.0000±0.0000 | 0.0000±0.0000 |
| yes | yes | 0.001 | 0.0600±0.3136 | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0600±0.3136 | 0.0200±0.1414 | 0.0000±0.0000 |
| yes | yes | 0.010 | | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 |
| yes | yes | 0.100 | | 0.0000±0.0000 | 0.0200±0.1414 | 0.0000±0.0000 | 0.0000±0.0000 | 0.0000±0.0000 | 0.1000±0.3642 | 0.0000±0.0000 | 0.0600±0.2399 |

Table S VI: Percentage over 50 runs where the global optimum is found by GA for static and dynamic instances of the 0-1 knapsack problem with $l = 300$.

| | | | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 30 | 40 | 38 | 12 | 46 | 24 | 24 | 24 | 14 |
| no | no | 0.010 | | 2 | 36 | 30 | 12 | 34 | 6 | 28 | 26 |
| no | no | 0.100 | | 2 | 0 | 6 | 0 | 30 | 20 | 16 | 18 |
| no | yes | 0.001 | 26 | 18 | 20 | 28 | 6 | 50 | 10 | 38 | 18 |
| no | yes | 0.010 | | 4 | 40 | 56 | 18 | 22 | 20 | 26 | 12 |
| no | yes | 0.100 | | 10 | 30 | 34 | 10 | 34 | 22 | 34 | 40 |
| yes | no | 0.001 | 30 | 44 | 18 | 36 | 20 | 36 | 8 | 30 | 6 |
| yes | no | 0.010 | | 2 | 6 | 26 | 44 | 22 | 6 | 42 | 28 |
| yes | no | 0.100 | | 0 | 4 | 34 | 14 | 32 | 10 | 44 | 18 |
| yes | yes | 0.001 | 14 | 42 | 18 | 46 | 42 | 34 | 16 | 16 | 44 |
| yes | yes | 0.010 | | 32 | 40 | 28 | 52 | 24 | 24 | 22 | 42 |
| yes | yes | 0.100 | | 30 | 8 | 20 | 14 | 52 | 22 | 36 | 16 |

Table S VII: Average error (over 50 runs) for GA in static and dynamic instances of the 0-1 knapsack problem with $l = 300$. The result for the static instance is compared to the results for dynamic instances of different types, all with $\tau = 500$. The duration of each run for static and dynamic instances is equal ($l$ seconds).

| | | | DOP Type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RI | MI | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| no | no | 0.001 | 1.6200±1.6399 | 1.0800±1.0467 | 1.1600±1.0947 | 2.7400±1.8385(s) | 1.0600±1.2357 | 1.6200±1.3536 | 2.4800±2.0627(s) | 2.3600±2.1358 | 2.1000±1.3286 |
| no | no | 0.010 | | 4.7000±2.2790(s) | 1.5800±1.6174 | 1.3000±1.2164 | 2.5400±1.6189(s) | 1.3200±1.1683 | 4.1400±2.7181(s) | 1.5400±1.3584 | 1.4400±1.2149 |
| no | no | 0.100 | | 6.8800±3.4025(s) | 13.2200±3.4834(s) | 3.5400±2.0123(s) | 6.9800±3.0605(s) | 1.5400±1.5147 | 3.1200±2.8546(s) | 2.4400±1.6308(s) | 2.0600±1.5039 |
| no | yes | 0.001 | 1.9600±1.8948 | 2.7800±2.5737 | 2.4600±1.8649(s) | 1.6800±1.6591 | 2.8600±2.2858(s) | 0.9400±1.2191(s) | 3.7200±2.8717(s) | 1.6200±1.6523 | 2.6800±2.0146(s) |
| no | yes | 0.010 | | 3.7400±2.5137(s) | 1.1400±1.4984(s) | 0.8600±1.2291(s) | 2.2800±1.8521 | 1.9400±1.6464 | 2.5000±1.7409 | 2.0800±1.8165 | 2.9600±2.1567(s) |
| no | yes | 0.100 | | 3.2000±2.7994(s) | 2.3400±2.2188 | 1.2400±1.2545(s) | 3.9200±2.6866(s) | 1.7800±1.6449 | 2.3200±1.9424 | 1.7000±1.6568 | 1.5600±1.6801 |
| yes | no | 0.001 | 1.7000±1.7291 | 0.9400±0.9982(s) | 2.7000±1.9717(s) | 1.4200±1.3107 | 2.2400±1.6107 | 1.3200±1.2196 | 4.8600±3.2764(s) | 1.6600±1.4654 | 2.5000±1.3286(s) |
| yes | no | 0.010 | | 5.1400±2.4827(s) | 3.7400±2.2571(s) | 1.3600±1.1021 | 1.0200±1.1156(s) | 1.6200±1.1229 | 4.3400±3.1727(s) | 1.1000±1.3286(s) | 1.5000±1.3590 |
| yes | no | 0.100 | | 13.3200±3.9042(s) | 5.9200±2.9056(s) | 1.6400±1.6383 | 2.6600±1.8802(s) | 1.5200±1.4741 | 4.3400±2.7598(s) | 0.9200±1.0270(s) | 2.0400±1.8066 |
| yes | yes | 0.001 | 3.3000±2.5495 | 1.3400±1.4086(s) | 2.0600±1.7660(s) | 0.9000±1.0546(s) | 1.2800±1.4574(s) | 1.5800±1.6793(s) | 2.4400±1.8969 | 2.2000±1.8844(s) | 1.1200±1.3649(s) |
| yes | yes | 0.010 | | 1.5400±1.5546(s) | 1.4000±1.4983(s) | 1.8200±1.6986(s) | 1.1600±1.4758(s) | 2.2400±2.0953(s) | 2.9000±2.6745 | 2.4000±1.9483(s) | 1.4000±1.6782(s) |
| yes | yes | 0.100 | | 2.3600±2.3014(s) | 4.0800±2.7909 | 2.5000±2.1969 | 3.7600±2.8754 | 0.8000±1.0102(s) | 2.6200±2.2578 | 1.3200±1.3468(s) | 2.8600±2.2132 |

Table S VIII: Average error (over 50 runs) for static and dynamic instances when (1+1)-ES is applied in the optimization of real functions with $l = 10$. When the problem changes, the change period ($\tau$) is equal to 1000 generations. The duration of each run for static and dynamic instances is equal ($l$ seconds). The results for the static instance correspond to the best results among the experiments with different kick sizes (0, 0.1, 0.2, 0.5, 1.0).

| | | | DOP Type | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| prob. | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| $f_6$ | 0.001 | 26.4±61.7 | 4.2±10.8(s) | 3.5±2.1 | 1.3±3.6(s) | 3.0±2.2 | 4.0±2.7 | 4.0±2.7 | 32.8±68.9(s) | 0.5±0.7(s) |
| | 0.010 | | 0.4±0.2(s) | 3.5±2.1 | 0.4±0.1(s) | 2.4±1.8 | 0.9±0.7(s) | 3.5±2.7 | 26.4±61.7(s) | 1.0±1.0(s) |
| | 0.100 | | 18.6±12.9 | 5.5±4.1 | 1.5±1.1 | 2.4±1.8 | 1.2±1.1(s) | 10.2±16.0 | 24.9±61.3 | 1.4±1.2(s) |
| | 0.200 | | 84.7±59.0(s) | 44.1±37.2(s) | 1.6±0.9 | 6.2±4.4 | 1.7±1.2(s) | 50.5±82.4(s) | 23.7±59.2(s) | 2.2±2.0 |
| | 0.500 | | 1572.7±1980.4(s) | 111.8±115.6(s) | 1.5±1.1(s) | 15.8±15.9 | 2.6±1.7 | 80089.7±353212.9(s) | 25.6±59.9 | 8.0±34.6 |
| | 0.900 | | 7326.3±7376.3(s) | 562.2±1091.1(s) | 4.0±2.4 | 48.9±33.6(s) | 7328142.9±50108826.8(s) | 41516724.5±121583920.9(s) | 24.5±58.8 | 256430.1±1496438.3(s) |
| $f_7$ | 0.001 | 4.6±2.2 | 316.4±466.8(s) | 0.4±0.2(s) | 0.1±0.0(s) | 0.3±0.2(s) | 0.2±0.1(s) | 0.2±0.1(s) | 35.4±28.2(s) | 34.5±114.1 |
| | 0.010 | | 311.1±472.5(s) | 0.4±0.2(s) | 0.3±0.4(s) | 0.3±0.2(s) | 1.7±10.3(s) | 44.1±100.4(s) | 35.4±28.2(s) | 36.2±113.9 |
| | 0.100 | | 234.9±372.7(s) | 0.7±0.3(s) | 1.8±5.2(s) | 0.3±0.2(s) | 11.7±21.7 | 304.0±458.6(s) | 35.4±28.2(s) | 47.1±103.0(s) |
| | 0.200 | | 262.6±283.1(s) | 0.9±0.4(s) | 10.2±34.5(s) | 13.7±69.7(s) | 108.5±357.6(s) | 331.3±455.5(s) | 34.4±28.4(s) | 75.9±148.9(s) |
| | 0.500 | | 393.7±438.0(s) | 1.1±0.3(s) | 130.7±358.3(s) | 7.8±13.8 | 177.9±391.6(s) | 370.9±476.7(s) | 33.8±25.2(s) | 117.1±230.1(s) |
| | 0.900 | | 412.3±470.8(s) | 1.3±0.4(s) | 206.4±384.6(s) | 88.1±216.9(s) | 249.0±387.7(s) | 410.3±474.0(s) | 35.1±28.2(s) | 181.1±275.3(s) |
| $f_8$ | 0.001 | 20.2±0.0 | 20.3±0.1(s) | 20.3±0.1(s) | 20.2±0.0 | 20.3±0.0(s) | 20.2±0.1 | 20.2±0.1 | 20.2±0.0 | 20.2±0.0 |
| | 0.010 | | 20.3±0.1(s) | 20.3±0.1(s) | 20.2±0.1 | 20.3±0.0(s) | 20.2±0.1 | 20.2±0.0(s) | 20.2±0.0 | 20.2±0.0(s) |
| | 0.100 | | 20.3±0.1(s) | 20.3±0.1(s) | 20.2±0.1 | 20.3±0.0(s) | 20.2±0.1 | 20.2±0.1 | 20.2±0.0 | 20.2±0.0 |
| | 0.200 | | 20.3±0.1(s) | 20.3±0.1(s) | 20.2±0.1 | 20.3±0.1(s) | 20.2±0.1 | 20.2±0.1 | 20.2±0.1 | 20.2±0.1(s) |
| | 0.500 | | 20.3±0.1(s) | 20.3±0.1(s) | 20.2±0.1 | 20.3±0.1(s) | 20.2±0.0 | 20.3±0.0(s) | 20.2±0.1(s) | 20.3±0.0(s) |
| | 0.900 | | 20.3±0.1(s) | 20.3±0.1(s) | 20.2±0.1 | 20.3±0.1(s) | 20.3±0.1(s) | 20.3±0.0(s) | 20.2±0.0 | 20.3±0.1(s) |
| $f_9$ | 0.001 | 16.7±4.4 | 16.9±11.9 | 1.7±0.9(s) | 16.5±11.7 | 1.8±1.2(s) | 2.5±0.9(s) | 2.5±0.9(s) | 29.5±18.6(s) | 2.5±0.8(s) |
| | 0.010 | | 0.5±0.6(s) | 1.7±0.9(s) | 1.5±1.0(s) | 1.8±1.2(s) | 3.6±0.8(s) | 7.8±2.2(s) | 29.5±18.6(s) | 4.5±1.5(s) |
| | 0.100 | | 2.6±0.8(s) | 2.7±1.1(s) | 2.7±0.8(s) | 1.8±1.2(s) | 3.4±1.2(s) | 13.5±4.0(s) | 29.5±18.6(s) | 4.8±1.5(s) |
| | 0.200 | | 3.1±0.9(s) | 3.3±0.8(s) | 2.8±1.0(s) | 2.2±1.1(s) | 3.7±1.0(s) | 18.1±6.0 | 28.5±17.6(s) | 5.7±2.3(s) |
| | 0.500 | | 6.9±2.1(s) | 4.2±1.3(s) | 4.0±1.1(s) | 2.5±1.0(s) | 4.8±1.7(s) | 32.1±12.1(s) | 29.1±19.9(s) | 7.7±3.0(s) |
| | 0.900 | | 10.6±2.9(s) | 4.6±1.7(s) | 5.4±1.8(s) | 3.2±0.9(s) | 17.7±12.2 | 61.3±17.5(s) | 30.3±17.6(s) | 21.9±13.5(s) |
| $f_{10}$ | 0.001 | 36.4±9.5 | 46.1±32.2 | 7.0±2.4(s) | 43.8±30.3 | 7.9±2.6(s) | 7.1±1.6(s) | 7.1±1.6(s) | 45.1±32.4 | 8.5±2.2(s) |
| | 0.010 | | 31.9±17.3 | 7.0±2.4(s) | 35.3±24.9 | 7.9±2.6(s) | 7.8±2.0(s) | 15.0±4.5(s) | 45.1±32.4 | 10.7±3.5(s) |
| | 0.100 | | 6.7±2.1(s) | 8.8±3.4(s) | 6.6±2.2(s) | 7.9±2.6(s) | 8.8±2.4(s) | 28.3±7.0(s) | 45.1±32.4 | 12.0±4.4(s) |
| | 0.200 | | 7.3±2.5(s) | 9.9±3.3(s) | 7.4±1.9(s) | 8.8±2.4(s) | 9.6±2.8(s) | 36.6±9.5 | 44.7±32.0 | 13.4±4.2(s) |
| | 0.500 | | 19.2±7.3(s) | 12.2±4.3(s) | 9.4±2.5(s) | 9.0±3.0(s) | 12.1±3.2(s) | 53.0±14.8(s) | 42.6±32.2 | 22.2±12.2(s) |
| | 0.900 | | 40.8±10.5(s) | 15.5±4.8(s) | 19.4±5.5(s) | 11.6±3.4(s) | 44.7±22.6(s) | 91.6±26.3(s) | 44.9±32.0 | 49.1±21.9(s) |
| $f_{11}$ | 0.001 | 4.7±1.7 | 4.8±1.8 | 5.2±1.2 | 4.7±1.5 | 4.7±0.8 | 4.2±0.9 | 4.2±0.9 | 4.7±1.7 | 4.3±1.3(s) |
| | 0.010 | | 3.9±1.1(s) | 5.2±1.2 | 4.0±1.2(s) | 4.7±0.8 | 4.3±1.4 | 5.3±0.9(s) | 4.7±1.7 | 4.4±1.0 |
| | 0.100 | | 5.2±1.4 | 6.0±1.4(s) | 4.2±1.1 | 4.7±0.8 | 4.4±0.8 | 5.8±0.9(s) | 4.7±1.6 | 5.0±1.1 |
| | 0.200 | | 6.2±1.2(s) | 6.0±1.2(s) | 4.7±1.4 | 5.8±1.1(s) | 4.6±1.1 | 6.4±0.9(s) | 4.9±1.6 | 5.8±1.5(s) |
| | 0.500 | | 6.4±1.5(s) | 6.1±1.4(s) | 4.4±1.2 | 6.1±1.1(s) | 4.7±1.1 | 7.5±1.2(s) | 4.8±1.6 | 6.1±1.5(s) |
| | 0.900 | | 6.1±1.6(s) | 6.5±1.6(s) | 5.1±1.5 | 6.2±1.2(s) | 5.9±1.8(s) | 8.1±1.3(s) | 4.5±1.2 | 7.0±1.6(s) |
| $f_{12}$ | 0.001 | 92.3±62.4 | 3635.6±5323.4(s) | 16.9±6.7(s) | 6180.9±10394.4(s) | 15.9±7.6(s) | 68.4±232.6(s) | 68.4±232.6(s) | 7273.8±12372.6(s) | 12.1±10.0(s) |
| | 0.010 | | 358.8±708.9 | 16.9±6.7(s) | 1315.9±5071.1 | 15.9±7.6(s) | 24.5±100.8(s) | 54.7±30.9(s) | 7273.3±12372.9(s) | 14.1±7.0(s) |
| | 0.100 | | 15.8±4.9(s) | 26.6±11.4(s) | 34.6±109.4(s) | 15.9±7.6(s) | 12.8±7.4(s) | 138.1±143.6 | 7273.8±12372.6(s) | 19.0±13.3(s) |
| | 0.200 | | 26.5±7.4(s) | 46.4±38.9(s) | 10.5±3.5(s) | 25.6±11.7(s) | 15.0±7.9(s) | 458.5±636.9(s) | 7273.4±12372.8(s) | 19.9±10.6(s) |
| | 0.500 | | 82.5±52.2 | 49.0±27.5(s) | 14.7±6.4(s) | 31.5±15.7(s) | 14.2±7.4(s) | 1365.3±1495.8(s) | 7235.4±12392.8(s) | 35.3±40.3(s) |
| | 0.900 | | 220.3±210.0(s) | 86.7±61.1 | 18.9±10.4(s) | 43.0±21.0(s) | 32.3±34.4(s) | 10834.4±11505.9(s) | 7348.7±12083.9(s) | 152.3±338.7 |

Table S IX: Average error (over 50 runs) for static and dynamic instances when $(\mu, \lambda)$-ES is applied in the optimization of real functions with $l = 10$ and $l = 30$. When the problem changes, the change period ($\tau$) is equal to 300 generations. The duration of each run for static and dynamic instances is equal ($3l$ seconds).

| | | | | DOP Type | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| prob. | $l$ | $\rho$ | static | 1.1 | 1.2 | 1.3 | 2.1 | 2.2 | 2.3 | 3.1 | mixed |
| $f_6$ | 10 | 0.001 | 0.5±1.9 | 0.4±1.7(s) | 0.5±0.9(s) | 0.3±1.2(s) | 0.8±1.2(s) | 2.0±2.2(s) | 2.0±2.2(s) | 0.5±2.1 | 0.7±0.7(s) |
| | | 0.010 | | 0.2±0.5(s) | 0.5±0.9(s) | 0.4±0.7(s) | 0.8±1.2(s) | 0.8±1.3(s) | 1.7±2.2(s) | 0.5±2.1 | 0.8±1.1(s) |
| | | 0.100 | | 1.3±1.3(s) | 1.0±1.1(s) | 0.9±1.3(s) | 0.8±1.2(s) | 0.7±1.1(s) | 2.1±2.1(s) | 0.5±2.0 | 0.7±1.0(s) |
| $f_7$ | | 0.001 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) |
| | | 0.010 | | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) |
| | | 0.100 | | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) | 0.0±0.0(s) |
| $f_8$ | | 0.001 | 20.2±0.0 | 20.2±0.0(s) | 20.2±0.1 | 20.2±0.0(s) | 20.2±0.1 | 20.2±0.0(s) | 20.2±0.0(s) | 20.2±0.0(s) | 20.2±0.0(s) |
| | | 0.010 | | 20.2±0.0(s) | 20.2±0.1 | 20.2±0.0(s) | 20.2±0.1 | 20.2±0.0(s) | 20.2±0.0(s) | 20.2±0.0(s) | 20.2±0.0(s) |
| | | 0.100 | | 20.2±0.1(s) | 20.2±0.0(s) | 20.2±0.0(s) | 20.2±0.1 | 20.2±0.1(s) | 20.2±0.0(s) | 20.2±0.0(s) | 20.2±0.0(s) |
| $f_9$ | | 0.001 | 13.2±6.7 | 13.3±6.8(s) | 12.5±6.1(s) | 13.6±7.1(s) | 4.1±2.1(s) | 12.5±6.5(s) | 12.5±6.5(s) | 13.4±6.9(s) | 10.5±5.8(s) |
| | | 0.010 | | 13.5±7.0(s) | 12.5±6.1(s) | 13.4±7.0(s) | 4.1±2.1(s) | 9.4±4.2(s) | 9.7±4.0(s) | 13.5±6.9(s) | 10.2±4.5(s) |
| | | 0.100 | | 10.1±6.1(s) | 12.5±6.3 | 10.6±5.8(s) | 4.0±2.2(s) | 9.7±4.8(s) | 3.8±2.0(s) | 13.6±7.1(s) | 7.5±3.4(s) |
| $f_{10}$ | | 0.001 | 3.7±3.2 | 3.7±3.2(s) | 4.7±4.4(s) | 3.8±3.1(s) | 4.5±3.9(s) | 3.5±3.0 | 3.5±3.0 | 3.9±3.1(s) | 3.6±3.0(s) |
| | | 0.010 | | 3.8±3.2(s) | 4.7±4.4(s) | 3.7±3.3(s) | 4.5±3.9(s) | 3.5±3.0(s) | 0.3±0.4(s) | 3.9±3.1(s) | 1.8±1.2(s) |
| | | 0.100 | | 4.1±3.0 | 5.6±6.1(s) | 3.4±2.6 | 4.5±3.9(s) | 3.7±2.9(s) | 0.2±0.4(s) | 3.9±3.1(s) | 0.8±0.8(s) |
| $f_{11}$ | | 0.001 | 3.5±1.4 | 3.5±1.4(s) | 2.8±1.2(s) | 3.5±1.4 | 2.8±1.2(s) | 3.3±1.3 | 3.3±1.3 | 3.5±1.4(s) | 3.3±1.3 |
| | | 0.010 | | 3.4±1.4 | 2.8±1.2(s) | 3.4±1.3 | 2.8±1.2(s) | 2.6±1.1(s) | 2.4±1.0(s) | 3.5±1.4(s) | 3.1±1.4 |
| | | 0.100 | | 2.6±1.2(s) | 2.5±1.2(s) | 2.6±1.3(s) | 2.8±1.2(s) | 2.6±1.2(s) | 2.3±1.0(s) | 3.5±1.4(s) | 2.6±1.2(s) |
| $f_{12}$ | | 0.001 | 858.4±1564.3 | 890.9±1592.5 | 357.9±1206.1(s) | 814.2±1571.8(s) | 122.0±602.6(s) | 1109.2±1970.1(s) | 1109.3±1970.1(s) | 804.5±1525.4 | 708.7±1995.8 |
| | | 0.010 | | 821.1±1558.5(s) | 357.9±1206.1(s) | 862.6±1562.1(s) | 121.8±602.7(s) | 841.0±1530.8 | 319.0±973.4(s) | 804.5±1525.4 | 347.3±1153.4(s) |
| | | 0.100 | | 416.1±1129.7(s) | 119.2±676.4(s) | 780.7±1496.8(s) | 121.8±602.7(s) | 645.9±1264.4(s) | 7.7±9.2(s) | 804.5±1525.4 | 215.9±668.4(s) |
| $f_6$ | 30 | 0.001 | 4043.1±921.0 | 4914.2±1357.1(s) | 5950.4±1587.7(s) | 4610.2±1121.2(s) | 6377.0±2162.9(s) | 5781.1±1457.8(s) | 6102.8±1627.9(s) | 4517.0±1171.5(s) | 5362.5±1478.1(s) |
| | | 0.010 | | 5446.2±1428.6(s) | 5948.8±1585.9(s) | 5071.9±1308.3(s) | 6377.0±2162.9(s) | 5352.0±1340.3(s) | 6490.7±1822.7(s) | 4494.2±1149.5(s) | 5529.9±1307.3(s) |
| | | 0.100 | | 7289.7±1618.5(s) | 6096.9±1862.3(s) | 5452.6±1371.6(s) | 7034.4±1628.5(s) | 5693.8±1275.3(s) | 6530.5±1683.4(s) | 4382.2±1080.5(s) | 6586.5±1931.9(s) |
| $f_7$ | | 0.001 | 1.1±0.0 | 3.2±0.7(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.1±0.0(s) | 1.2±0.0(s) |
| | | 0.010 | | 2.9±0.6(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.1±0.0(s) | 1.2±0.0(s) |
| | | 0.100 | | 1.5±0.2(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.2±0.0(s) | 1.1±0.0(s) | 1.2±0.0(s) |
| $f_8$ | | 0.001 | 20.9±0.0 | 20.9±0.0(s) | 20.9±0.1(s) | 20.9±0.1 | 20.9±0.0(s) | 20.9±0.0(s) | 20.9±0.1 | 20.9±0.0(s) | 20.9±0.0(s) |
| | | 0.010 | | 20.9±0.0(s) | 20.9±0.1(s) | 20.9±0.0(s) | 20.9±0.0(s) | 20.9±0.0(s) | 20.9±0.1 | 20.9±0.0(s) | 20.9±0.0(s) |
| | | 0.100 | | 20.9±0.0(s) | 20.9±0.0(s) | 20.9±0.0 | 20.9±0.0(s) | 20.9±0.0(s) | 20.9±0.0 | 20.9±0.0(s) | 20.9±0.1 |
| $f_9$ | | 0.001 | 177.6±24.5 | 180.6±24.3(s) | 171.6±22.2(s) | 180.6±23.8(s) | 171.4±26.4(s) | 84.1±19.8(s) | 104.2±23.7(s) | 178.6±22.8 | 142.7±33.7(s) |
| | | 0.010 | | 179.6±25.5 | 171.6±22.2(s) | 179.2±23.9 | 171.4±26.4(s) | 69.2±14.7(s) | 91.5±22.0(s) | 179.0±22.7 | 133.1±31.6(s) |
| | | 0.100 | | 32.8±14.6(s) | 155.9±22.4(s) | 60.0±12.8(s) | 168.2±26.9(s) | 57.0±12.9(s) | 78.2±18.3(s) | 178.1±23.1 | 67.5±16.5(s) |
| $f_{10}$ | | 0.001 | 32.8±5.8 | 35.8±7.0(s) | 35.1±3.4(s) | 35.1±6.1(s) | 33.6±4.7 | 29.5±3.5(s) | 36.4±4.3(s) | 35.2±4.2(s) | 32.7±4.1 |
| | | 0.010 | | 37.5±6.5(s) | 35.1±3.4(s) | 35.3±5.9(s) | 33.6±4.7 | 29.8±3.1(s) | 36.2±4.7(s) | 35.2±4.2(s) | 33.2±3.9 |
| | | 0.100 | | 35.3±4.0(s) | 37.8±4.0(s) | 29.5±3.9(s) | 34.1±3.8 | 30.7±3.4(s) | 37.8±4.1(s) | 35.2±4.3(s) | 33.1±3.7 |
| $f_{11}$ | | 0.001 | 14.2±2.5 | 13.6±2.2 | 13.7±2.1 | 13.7±2.4 | 13.0±2.1(s) | 12.4±2.0(s) | 16.7±2.6(s) | 14.3±2.2 | 13.5±2.5(s) |
| | | 0.010 | | 13.7±2.5 | 13.7±2.1 | 13.9±2.5 | 13.0±2.1(s) | 12.1±2.1(s) | 17.8±2.3(s) | 14.3±2.2 | 13.9±2.8 |
| | | 0.100 | | 12.4±2.0(s) | 14.9±2.1 | 12.4±1.9(s) | 13.7±2.0 | 12.1±1.7(s) | 18.5±2.3(s) | 14.4±2.2 | 14.3±2.5 |
| $f_{12}$ | | 0.001 | 8942.8±7572.9 | 8724.4±7419.3 | 8793.5±6077.7 | 9651.3±8090.3(s) | 9550.1±7960.1(s) | 9243.0±8040.2 | 15526.9±11873.1(s) | 9102.0±8137.5 | 9167.2±7107.6 |
| | | 0.010 | | 9250.5±8910.1 | 8948.1±6080.7 | 9657.2±7938.4(s) | 8898.1±7821.7(s) | 10275.9±9025.3(s) | 12693.7±8188.4(s) | 8979.1±8059.6 | 9124.9±6879.5 |
| | | 0.100 | | 11082.1±8670.8(s) | 13616.2±8239.9(s) | 10136.9±7497.1(s) | 10986.9±6307.5(s) | 10074.5±8488.5(s) | 17594.0±7217.3(s) | 9010.0±7833.2 | 9896.1±6595.7(s) |

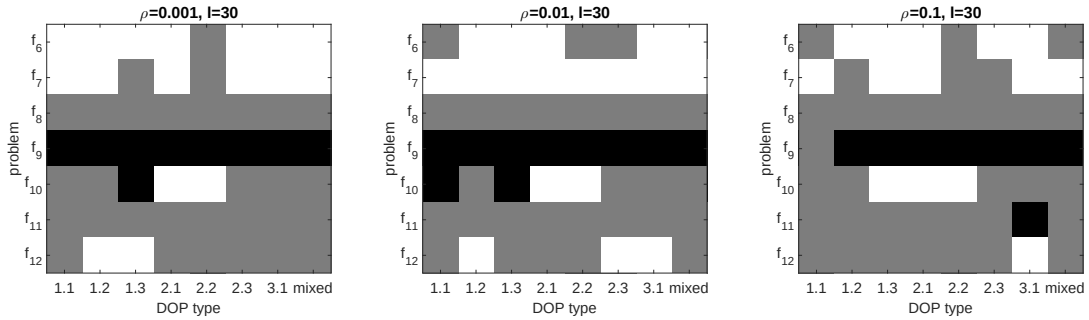


Figure S 1: Comparison of $(\mu, \lambda)$-ES with and without $mInc$ strategy in the optimization of continuous functions with $l = 30$, all with $\tau = 300$. When the statistical test indicates no difference between the results, a gray square is shown. Otherwise, a black or white square is shown when the result for $(\mu, \lambda)$-ES with $mInc$ is respectively better or worse than the result for $(\mu, \lambda)$-ES without $mInc$.

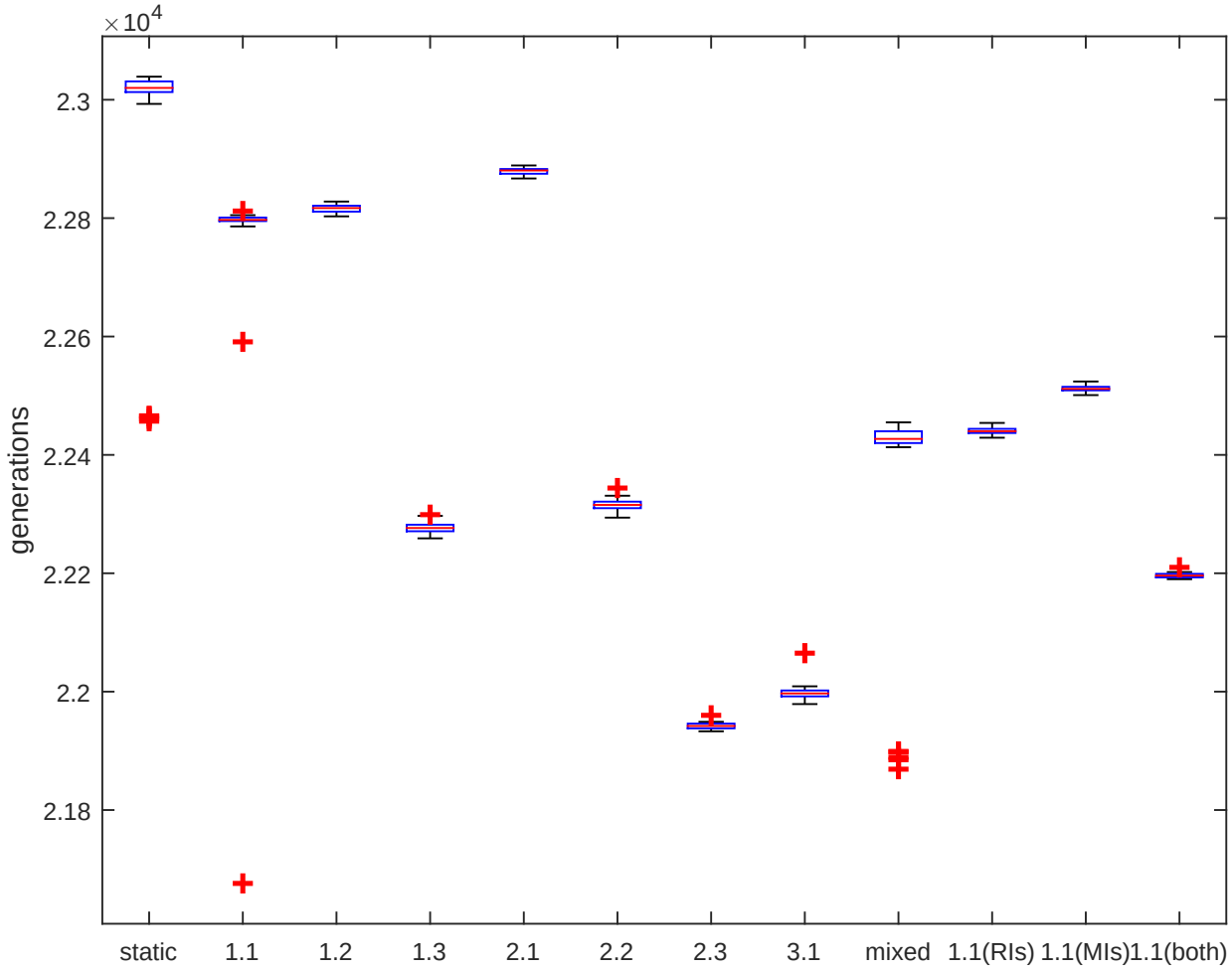


Figure S 2: Number of generations for the GA in static and dynamic instances of the 0-1 knapsack problem with $l = 200$. The result for the static instance is compared to the results for dynamic instances of different types, all with $\tau = 5$. The duration of each run for static and dynamic instances is equal ($l/10$ seconds). All the other parameters used for the GA are equal to those used in previous experiments. For the dynamic instance with Change Type 1.1, results for runs with RIs and MIs are also presented. As the runtime is fixed, the number of generations in a run gives information about the cost of applying the changes in the problem. One can observe that the number of generations for the static instances is higher than for the dynamic instances. However the difference in the number of generations is not large. Also, the number of generations is smaller for changes 1.3, 2.2, 2.3, and 3.1. For those changes, the modification in the fitness is applied only to a subset of solutions; finding this subset impacts the number of generations of the algorithm. Also, inserting immigrants impacts the number of generations; the impact is higher for the random immigrants approach.